MASTER IN COMPUTER SCIENCE PROFESSIONAL FOCUS IN SOFTWARE ENGINEERING

Parameter tuning of deep learning using evolutionary algorithm

Hendrix, Maxime

Award date:
2018

Awarding institution:
University of Namur

Université de Namur
Faculty of Computer Science
Academic Year 2017–2018

Parameter tuning of deep learning using evolutionary algorithm

Hendrix Maxime

Internship mentor: Divina Federico

Supervisor: _____ (Signed for Release Approval - Study Rules art. 40)
Wim VANHOOF

A thesis submitted in the partial fulfillment of the requirements
for the degree of Master of Computer Science at the Université of Namur

Abstract

The deep learning is an algorithm based on machine learning that allows to forecast the consumption of electricity in Spain. The deep learning algorithm receives input data that represent the past consumption of electricity. After different computations, the desired forecast is obtained as an output. However, the deep learning algorithm has some parameters that need to be configured in order to predict with accuracy. A Genetic algorithm is used to perform this parametrization and to find the optimal parameters.

The goal of this thesis is to optimize the parameters of deep learning algorithm in order to forecast with more accuracy the consumption of electricity in Spain. The results obtained by the deep learning algorithm with an optimization of the parameters are better than without. There are other methods that allow to forecast the electricity consumption. The comparison between our developed algorithm and the other techniques shows that our algorithm is more efficient to forecast the consumption of electricity.

keywords: machine learning, genetic algorithm, deep learning algorithm, forecasting, time series, optimization parameters

L'apprentissage profond est un algorithme basé sur l'apprentissage automatique qui permet de prédire la consommation électrique en Espagne. L'algorithme d'apprentissage profond reçoit des données en entrée qui représentent la consommation passée d'électricité. Après différents calculs, la prédiction attendue est obtenue en sortie de l'algorithme. Cependant, les paramètres de l'algorithme d'apprentissage profond doivent être configurés afin de prédire avec plus d'efficacité. Un algorithme génétique est utilisé pour effectuer cette paramétrisation et donc trouver les paramètres optimaux.

Le but de cette thèse est d'optimiser les paramètres d'un algorithme d'apprentissage profond afin de prédire avec le plus de précision la consommation électrique future en Espagne. Les résultats obtenus avec l'optimisation des paramètres sont meilleurs que ceux obtenus sans optimisation. Il existe d'autres méthodes qui permettent de prédire la consommation électrique. La comparaison entre notre algorithme développé et les autres techniques montre que notre algorithme est plus efficace.

mots-clefs: apprentissage automatique, algorithme génétique, algorithme d'apprentissage profond, prédiction, séries temporelles, optimisation des paramètres

Preface

This paper is submitted for the Master's Degree of Computer Sciences at the University of Namur. This research was conducted under the supervision of Professor Vanhoof of the university of Namur. The internship took place in the university of Pablo de Olavide in Seville with the assistant professor, Mister Federico Divina.

First of all, I would like to say thank to Mister Vanhoof to offer me the possibility to go away in Spain to do this study and for the different advices that he gave me during the realization of my thesis. I am particularly grateful for the devoted time to the careful reading of the thesis.

I would like to say thank to my internship mentor, Federico Divina. Every week, he took the time to receive me, to discuss about the thesis and the research.

In my research, I needed to reuse and to modify one algorithm that was already developed at the university. I would like to thank the people who helped me to understand this code and this algorithm, J. F. Torres and F. Martínez Álvarez.

I also thank the other people in the laboratory of Big Data in the university of Pablo de Olavide for their welcome and help during these three months of internship.

I thank to the CICA that is the center of computer sciences of Andalousia for allowing me to use their servers to execute my different experimentation during my research.

Finally, I thank my family and friends that helped me to realize this work and for their encouragements throughout this project.

Maxime HENDRIX

June 2018

Contents

Abstract 3

Preface 5

Glossary 15

1 Introduction 17

2 State of the art 21

2.1 Deep learning algorithms . . . 21

2.1.1 History of deep learning algorithms . . . 21

2.1.2 Research about deep learning algorithms . . . 22

2.2 Evolutionary algorithms . . . 24

2.2.1 History of evolutionary algorithms . . . 24

2.2.2 Research about evolutionary algorithms . . . 25

2.3 Energy forecasting . . . 26

3 Deep Learning algorithms 29

3.1 Deep learning algorithms process . . . 29

3.1.1 Input layer . . . 30

3.1.2 Hidden layer . . . 30

3.1.2.1 Neurons . . . 30

3.1.2.2 Synapses . . . 31

3.1.3 Output layer . . . 31

3.1.4 Learning process . . . 31

3.2 Parameters to optimize . . . 32

3.2.1 Number of hidden layers . . . 32

3.2.2 Number of neurons per hidden layer . . . 33
3.2.3 Value of lambda . . . 33
3.2.4 Value of rho . . . 34
3.2.5 Value of epsilon . . . 34
3.2.6 Activation function . . . 35
3.2.7 Distribution function . . . 35
3.2.8 End metric . . . 36
3.3 Execution time of deep learning . . . 37
3.4 Parameter summary . . . 38

4 Genetic algorithms 39
4.1 The genetic algorithms' process . . . 39
4.1.1 Initialisation . . . 40
4.1.2 Parent selection . . . 40
4.1.3 Genetic operators . . . 42
4.1.4 Survivor selection . . . 43
4.1.5 Termination . . . 43
4.2 Parameters of the genetic algorithm . . . 44
4.2.1 Type of genetic algorithm . . . 44
4.2.2 Population size . . . 44
4.2.3 Number of generations . . . 45
4.2.4 Limit of generations . . . 45
4.2.5 Percentage of crossover . . . 46
4.2.6 Percentage of elitism . . . 46
4.2.7 Percentage of mutation . . . 47
4.2.8 Local optimization . . . 47
4.2.9 Selection function . . . 47
4.2.10 Population function . . . 48
4.2.11 Crossover function . . . 48
4.2.12 Mutation function . . . 49
4.2.13 Parameter summary . . . 49

5 Results 51
5.1 Package's versions . . . 51

5.2 Used data . . . 52
5.3 Mean relative error . . . 53
5.4 Analysis of the results . . . 54
5.4.1 Results of deep learning without optimization . . . 54
5.4.1.1 Parameters of the deep learning . . . 54
5.4.1.2 Result of MRE for the 24 subproblems . . . 54
5.4.2 Results of deep learning with a global optimization . . . 56
5.4.2.1 Parameters of the deep learning . . . 56
5.4.2.2 Result of MRE for the 24 subproblems . . . 57
5.4.3 Results of deep learning with a specific optimization . . . 58
5.4.3.1 Parameters of the deep learning . . . 58
5.4.3.2 Result of MRE for the 24 subproblems . . . 61
5.4.4 Comparison between the 3 techniques . . . 62
5.5 Comparison with other techniques . . . 63

6 Conclusion **67**

Bibliographie **73**

Appendices **73**

A Evolution of the genetic algorithm for the subproblems **75**

List of Figures

3.1 The process of deep learning algorithms [13] 29

3.2 The representation of one neuron [7] 31

4.1 The process of genetic algorithms [1] 39

5.1 Explanation of the forecasting [41] 53

5.2 Evolution of genetic algorithm for the subproblem 1 62

5.3 Evolution of the MRE in % according to the historical window for the different techniques 65

A.1 Evolution of genetic algorithm for the subproblem 1 75

A.2 Evolution of genetic algorithm for the subproblem 2 76

A.3 Evolution of genetic algorithm for the subproblem 3 76

A.4 Evolution of genetic algorithm for the subproblem 4 77

A.5 Evolution of genetic algorithm for the subproblem 5 77

A.6 Evolution of genetic algorithm for the subproblem 6 78

A.7 Evolution of genetic algorithm for the subproblem 7 78

A.8 Evolution of genetic algorithm for the subproblem 8 79

A.9 Evolution of genetic algorithm for the subproblem 9 79

A.10 Evolution of genetic algorithm for the subproblem 10 80

A.11 Evolution of genetic algorithm for the subproblem 11 80

A.12 Evolution of genetic algorithm for the subproblem 12 81

A.13 Evolution of genetic algorithm for the subproblem 13 81

A.14 Evolution of genetic algorithm for the subproblem 14 82

A.15 Evolution of genetic algorithm for the subproblem 15 82

A.16 Evolution of genetic algorithm for the subproblem 16 83

A.17 Evolution of genetic algorithm for the subproblem 17 83
A.18 Evolution of genetic algorithm for the subproblem 18 84
A.19 Evolution of genetic algorithm for the subproblem 19 84
A.20 Evolution of genetic algorithm for the subproblem 20 85
A.21 Evolution of genetic algorithm for the subproblem 21 85
A.22 Evolution of genetic algorithm for the subproblem 22 86
A.23 Evolution of genetic algorithm for the subproblem 23 86
A.24 Evolution of genetic algorithm for the subproblem 24 87

List of Tables

2.1 Metaphor between evolution and problem solving 24

3.1 Execution time of deep learning algorithm (in seconds) 37
3.2 Different possible value for the numeric parameters 38

4.1 Results with different population size 45
4.2 Results with different generations 45
4.3 Results with different percentage of crossover 46
4.4 Results with different percentage of elitism 46
4.5 Results with different percentage of mutation 47
4.6 Results with different selection functions 48
4.7 Results with different crossover functions 48
4.8 Results with different mutation functions 49
4.9 Parameters used for the genetic algorithm 49

5.1 The different versions of the used package 52
5.2 Parameter used for the deep learning algorithm without optimization 54
5.3 Values of MRE in % without optimization for the 24 subproblems 55
5.4 Parameters found for the deep learning algorithm with a global optimization . . 56
5.5 Values of MRE in % with a global optimization for the 24 subproblems 57
5.6 Parameters found for the deep learning algorithm with a specific optimization . . 59
5.7 Values of MRE in % with a specific optimization for the 24 subproblems 61
5.8 MRE in % for the 3 different techniques of deep learning 63
5.9 MRE in % for different forecasting techniques 64

Glossary

The definitions were found on techopedia's website [9]. The different definitions present and introduce the different concepts that are discussed on this thesis.

Artificial Intelligence : artificial intelligence is an area of computer science that emphasizes the creation of intelligent machines that work and react like humans. Some of the activities computers with artificial intelligence are designed for include: speech recognition, learning, planning and problem solving.

Big Data : big data refers to a process that is used when traditional data mining and handling techniques cannot uncover the insights and meaning of the underlying data. Data that is unstructured or time sensitive or simply very large cannot be processed by relational database engines. This type of data requires a different processing approach called big data, which uses massive parallelism on readily-available hardware.

Machine Learning : machine learning is a type of artificial intelligence that allows software applications to become more accurate in predicting outcomes without being explicitly programmed. The basic premise of machine learning is to build algorithms that can receive input data and use statistical analysis to predict an output value within an acceptable range.

Deep Learning : deep learning is a collection of algorithms used in machine learning, used to model high-level abstractions in data through the use of model architectures, which are composed of multiple nonlinear transformations. It is part of a broad family of methods used for machine learning that are based on learning representations of data;

Evolutionary algorithm : An evolutionary algorithm is considered a component of evolutionary computation in artificial intelligence. An evolutionary algorithm functions through the selection process in which the least fit members of the population set are eliminated, whereas the fit members are allowed to survive and continue until better solutions are determined. In other words, evolutionary algorithms are computer applications which mimic biological processes in order to solve complex problems. Over time, the successful members evolve to present the optimized solution to the problem;

Genetic algorithm : A genetic algorithm is a heuristic search method used in artificial intelligence and computing. It is used for finding optimized solutions to search problems based on the theory of natural selection and evolutionary biology. Genetic algorithms

are excellent for searching through large and complex data sets. They are considered capable of finding reasonable solutions to complex issues as they are highly capable of solving unconstrained and constrained optimization issues;

Evolutionary Computation : Evolutionary computation is an artificial intelligence sub-field and closely linked to computational intelligence, involving lots of combinatorial optimization problems and continuous optimization. It is employed in problem-solving systems that use computational models with evolutionary processes as the key design elements. It is an abstraction from the evolutionary concept in biology since it deals with methods and concepts that are continually and selectively evolving and optimizing.

Time series : A time series is a sequence of values used to show the evolution (growth or decrease) of a quantity according to an elapsed time.

Chapter 1

Introduction

With evolution in computer performances, more and more data, also called big data, are generated and stored, available for future use or analysis. Data are obtained in many different ways, such as sensors, consumption surveys, polls and other techniques. The problem with this kind of data is that there are too many of them to be addressed by usual data mining tools. In most cases, it is ineffective or it takes too much time to process the data. It is thus necessary to develop a certain number of techniques and algorithms to be able to work with these data in order to manipulate them and to analyse them efficiently [33].

In this paper, time series data are used to carry out the different tasks and experiments. A time series is a sequence of values used to show the evolution (growth or decrease) of a quantity according to an elapsed time. For our work, the considered time series is related to the electricity consumption in Spain, expressed in megawatt (MW), from January 2007 to June 2016 which represents a large amount of data, around 500,000 stored measures available for processing and analysis. One measure is the consumption of electricity in Spain at a specific time over the period under consideration with a high sampling frequency (10 minutes). These data are saved in a CSV file of 23.9 MB. These data are divided into 2 different parts, 70% of the data are used for training and the rest is used for testing. In the 70% of the training data, 30% are also used for validation purpose.

In this study, the objective is to predict the continuation of the temporal sequence, in particular for the next 24 hours after a given historical window that represents a series of successive measurements. The size of the historical window, represented by w, is a parameter of the system. We define 24 subproblems, represented by H_i (for $1 \leq i \leq 24$), where H_i represents the problem of estimating the electricity consumption at the i^{th} hour after the given historical window. For example, the seventh subproblem H_7 represents the estimation of the electricity consumption at the 7^{th} hour after the historical window. Together, the set of all 24 subproblems represents the problem of forecasting the electricity consumption during a full day after the historical window.

The prediction of future data is something actually very important in some fields such as economy and environment. For example, participants in the electricity sector (demand and supply) are particularly interested in this area of research, which allows them to forecast future revenues, inventory management or control of various power plans [22] with a margin of error that may be more or less important depending on the use. As another example, the electricity consumption of buildings in China in 2011 accounted for 28% of the country's total electricity consumption. In the United States, the electricity consumption of buildings represents around 39% of the total energy consumption [8]. In view of these numbers, there are opportunities to reduce building electricity consumption. Time series forecasting can be used to try to help the infrastructure manager to make better decisions or to provide some relevant information (to control all kinds of equipment, to know what consumes the most/least and other kinds of decisions/information).

The technique used to make the prediction is deep learning, a method based on machine learning and on artificial intelligence. Deep learning takes an input X to return a result output Y. In our context, we provide a time series as an input and we expect a meaningful forecasting as an output. Deep learning involves the creation and the training of neuronal networks that are based on multi-layers. Each layer has some neurons and the number of neurons on each layer can be different. Neurons take as input the results that have been calculated by the neurons of the previous layer. In the case of the first layer, the input data will be used. The neurons in the neural network are connected to each other with different connections that are called 'synapses'. One synapse has a weight that is determined by the learning phase. The calculated result from a neuron on a previous layer is multiplied by the synaptic weight. The result of the multiplication is transmitted to a neuron on the next layer [28].

Deep learning is chosen because it is a technique that gives the best result in terms of error rate when compared to other methods (decision trees, linear regression, gradient-boosted and random forest)[8]. The problem of deep learning is that the execution is slow. This is dependent on the number of layers and on the number of neurons per layer. Indeed, a small network (few layers and few neurons per layer) has a fast execution, while a large network (a lot of layers and a lot of neurons per layer) has a slow execution.

Another issue of deep learning algorithms is the necessity to take into account different functions such as activation, distribution and other functions that are used by neurons to calculate the result. Indeed, deep learning algorithms are very sensitive to the initialisation and much attention must be given to this stage because it is necessary to correctly fix different parameters (number of neurons, number of layers, activation function and other parameters) that are important in order to obtain a good forecasting [41]. The parameters of deep learning algorithms can take different kind of values such as string, integer and real values.

There are three main kinds of techniques for fixing most optimal parameters (in this case, the deep learning parameters): exhaustive search [29], random search [3], or the use of an evolutionary algorithm [1]. An evolutionary algorithm is preferred and is used to try to optimize the various parameters to reach a forecast as close as possible to actuality. This technique is chosen in our work because it is an algorithm that allows reaching

quickly a solution with a good result compared to exhaustive search or to random search. Indeed, random search and exhaustive search must test a lot of possible combinations, which will take a long time when dealing with a lot of different parameters [3, 29]. Another reason is that the deep learning algorithm on its own takes already a lot of time to be executed. However, it is important to remember that evolutionary algorithms allow to approximate the best solution but does not guarantee to produce the best solution.

Evolutionary algorithms are based on the evolutionary principle: in a given population, the best elements survive, and the weakest are eliminated. In this study, genetic algorithms that are one subclass of evolutionary algorithms are preferred to optimize the parameters of the deep learning algorithm. Genetic algorithms consist in taking a set of initial values that are called the initial population. This population is evaluated by a function called fitness function that allows measuring the quality of each element. After the evaluation by the fitness function, the best values (the values that gave the best result by the evaluation by the fitness function) are retained to create another set of values (another population)[1]. There are two kinds of techniques to create a new population from retained values : mutation and crossover.

1. Mutation consists in alteration of one value to create another value in order to keep a diversity between different populations.
2. Crossover consists in taking two values of the parent population to create a single new value in the new population by combining characteristics of the two elements into one.

After having computed a set of mutations and crossovers, a new population is ready to be evaluated by the fitness function. These steps are repeated during a certain number of generations that are also called iterations. At the end of the genetic algorithm execution, the best set of value for the different parameters is obtained [38].

In this study, the population is one set of values for the different parameters of the deep learning algorithm. The fitness function is the algorithm of deep learning itself. The quality of each parameter value set is shown by the mean relative error (MRE) [41].

The MRE represents the error rate that was calculated between the forecasted and the actual values. The forecasted values are the calculated values by the execution of the deep learning algorithm. The actual values are the values that have actually been observed so it is the values that are in the 30% of testing data. The calculated MRE is also used to analyse the different results obtained in our study.

Analyses and comparisons of results have been divided into two distinct parts to facilitate understanding and discussions. Firstly, the results are compared between three different methods to see if it is actually interesting to mix the genetic algorithm and the deep learning algorithm:

1. the first method consists in using the deep learning algorithm with the basic parameters defined in [41]. This method is also called the default case;

2. the second method consists in using the deep learning algorithm together with a genetic algorithm in order to make a global optimization of the parameters. It means that the parameters of the deep learning algorithm are the same for every subproblem;

3. the third method consists in using the deep learning algorithm together with a genetic algorithm in order to make a specific optimization of the parameters. It means that the parameters of the deep learning algorithm can be different for every subproblem.

Secondly, the results obtained by deep learning without optimization and deep learning with genetic algorithm in order to make a specific optimization of the parameters are compared with other methods used for the prediction (linear regression, decision tree, gradient-boosted and random forest). The comparison indicates if the developed algorithm in our work is more efficient than the other techniques that are already used to forecast electricity consumption.

In summary, this work introduces an algorithm to forecast big data time series based on deep learning architectures combined with a genetic algorithm to optimize parameters. The goal is to reduce the error rate (MRE) as much as possible.

The remainder of this paper is structured as follows. The state of the art is reviewed and discussed in chapter 2. The deep learning algorithm and the various parameters to be optimized are introduced in chapter 3. The genetic algorithm is presented in chapter 4. Results and comparison with other techniques are reported and discussed in chapter 5. Finally, the drawn conclusions and the possible future works from this experiment are summarized in chapter 6.

Chapter 2

State of the art

In this chapter, different studies about deep learning algorithms and evolutionary algorithms are summarized. Moreover, different works in the field of energy forecasting are discussed. This study uses data about electricity consumption so special attention is paid to related work in this context.

2.1 Deep learning algorithms

This section is divided into two different parts. The first part is the history of deep learning algorithms. The second part is about different articles where deep learning algorithms are used and compared with other kinds of techniques.

2.1.1 History of deep learning algorithms

The history of deep learning algorithms is based on [37].

It is around 1950 that two neurologists developed the first neural network. This neural network is called a standard neural network (NN). The neural network needs input data to be computed. This network is composed of one input layer, one hidden layer and one output layer. These layers have a sequence of connected processors called neurons. These neurons can be activated or not. If they are activated, they sent the calculated result based on the input data to the next neurons. This network allows to transform input to output by following some rules. In 1965, the neural network can be trained by itself. Moreover, the network can be composed with more than one hidden layer. The connections between the neurons are called synapses and they are a weight on each synapse. This weight can influence the calculated result by the neurons. The neural network needs a training set to be trained by the use of a regression analysis. The training allows to modify the weight of synapses. In a master thesis around 1970, this is the first time where someone speaks about the backpropagation (BP). The BP is used

to minimize cost functions by adapting control parameters in order to produce better output.

By the late 1980s, researchers did a significant progress in the field of neural networks. At that time, it was observed that it is not necessary to have a lot of layers in a neural network to be close of the expected result. Additional hidden layers often did not seem to offer empirical benefits. It is in 2000s that the deep learning approach appears although the term had already been used in 1986. This is due to the convergence of three factors: the mass of data to be processed, the growing power of computers and the development of neural networks. Indeed, basic neural networks are not powerful enough to process a large amount of data. So, it is necessary for the researchers to develop new techniques to process and to analyse larger amounts of data. The term deep learning regroups a set of different algorithms based on machine learning. The most known kind of deep learning algorithms is deep neural networks. Deep learning algorithms and neural networks have led to great advances, for example in the field of facial recognition and speech recognition [6].

The process of deep learning algorithms is detailed in Chapter 3.

2.1.2 Research about deep learning algorithms

Actually, deep learning algorithms can be used in different fields of research as presented in the following papers.

Deep learning algorithms can be used in the field of medicine in order to determine if a patient has Alzheimer's disease. Indeed, [31] presents a new algorithm based on deep learning approach in order to diagnose with more efficiency and accuracy a disease of Alzheimer. The deep learning algorithm is composed of different layers and the final prediction is based on a voting schema. The authors created two different deep learning algorithms and four different voting schemas to compare them. The different results show that the architecture of deep learning algorithm is based on deep belief networks (DBN) and the voting schema is based on support vector machines (SVM). DBN is a kind of deep learning algorithm that creates a graphical model. SVM techniques are used to solve different problems of regression. The algorithm developed by the authors based on the DBN architecture and SVM voter can diagnose an Alzheimer disease with an accuracy of 90%. This article shows that the deep learning approach is effective to classify different data in order to determine if a patient has Alzheimer's disease.

Other articles in the field of medicine research are dealing with deep learning algorithms. In [14], the authors develop a deep learning algorithm for automated detection of diabetic retinopathy and diabetic macular edema in retinal fundus photographs. Another article [18] presents deep learning algorithms has a way to analyse the medical images with more efficiency.

Deep learning algorithms are also used in the field of environmental research. For example, the authors of [43] have developed a new algorithm based on deep learning

algorithms able to determine the quality of air in a city. The algorithms that are actually used to forecast the air quality are based on shallow models. It means neural networks with only 3 layers (one input layer, one hidden layer and one output layer). The presented algorithm in this paper is based on spatio-temporal deep learning (STDL) algorithm able to create a correlation between spatial and temporal. The proposed algorithm is compared with other techniques (STANN, ARMA and SVR models) to forecast the air quality. The result of this study shows that the STDL is the most efficient to forecast the air quality. The developed algorithm based on deep learning approach can predict the air quality with a mean absolute error of 9%.

Still in the field of environmental research, in [11], the authors have used a deep learning algorithm in order to predict the status of pro-environmental consumption. Another work [16] is about the creation of a neural network in order to predict an excavator's hourly energy consumption and CO2 emissions under different site conditions.

Another field where deep learning algorithms are used is the energy consumption. For example, in [17], the authors have used the deep learning approach in order to reduce the loss of energy by different buildings. The authors use three different kinds of deep learning algorithms. The first one is based on a recurrent neural network called long short-term memory (LSTM). The second one is based on a denoising autoencoders and the last one is based on a network which regresses the start time, end time and averages power demand of each appliance activation. The authors use seven different metrics (recall, precision, F1, accuracy, relative error in total energy, mean absolute error and proportion of total energy correctly assigned) in order to determine the performance of each developed algorithm. The calculated results are based on different devices such as: fridge, microwave and other devices. The three developed algorithms are compared with two other techniques (combinatorial optimization and factorial HMM). The most efficient technique is the LSTM. Indeed, LSTM has the best results in 5 metrics, but the comparison is not actually fair because the different algorithms need to be parametrize with more or less parameters. The chosen parametrization may influence the obtained results. Finally, the LSTM can predict the loss of energy with an accuracy of 5%.

Other articles in the field of energy consumption are based on deep learning algorithms. In [23], the authors have developed a deep learning approach in order to forecast the electricity current intensity of the air conditioning equipment. Another article [41] presents a deep learning algorithm in order to forecast the consumption of electricity in Spain. This paper is more detailed in the part of energy forecasting.

2.2 Evolutionary algorithms

This section is divided into two different parts. The first part is the history of evolutionary algorithms. The second part is about different articles where evolutionary algorithms are used to solve complex problems.

2.2.1 History of evolutionary algorithms

The history of evolutionary algorithms is based on [10] and on [1].

Evolutionary algorithms have been developed to solve different problems of optimization. This kind of algorithms is based on the biological evolution. Indeed, in a given population, the weakest individuals are eliminated and the best individuals are kept. In the case of evolutionary algorithms, it is the quality of an element (the capacity of the element to solve the problem) that determines the chance of the element to be kept. The following table shows the link between the biologic evolution and the evolutionary algorithms :

Evolution	Problem solving
Environment	Problem
Individual	Candidate solutions
Fitness	Quality

Table 2.1: Metaphor between evolution and problem solving

The first description of an evolutionary algorithm appears in 1958. The evolutionary algorithm was used to find a program that calculates a given input–output function. In the early 1960s, this is the first time when someone speaks about genetic algorithms in a paper. Genetic algorithms are one kind of evolutionary algorithms that are one sort of evolutionary computation. Genetic algorithms are the most used form of evolutionary algorithms. The evolutionary algorithms and genetic algorithms have continued to be developed and improved to solve more complex problems. The period from 1990 until 1997 is characterized by a lot of conferences and development about genetic algorithms. In the 2000s, evolutionary algorithms were strongly developed because the computer power was strongly increased and so, problems became more complex. Nowadays, some mathematical problems and optimization problems require the use of evolutionary algorithms in order to find quickly a good solution.

The process of genetic algorithms is detailed in Chapter 4.

2.2.2 Research about evolutionary algorithms

Actually, evolutionary algorithms are used in complex optimization problems.

In [26], the authors present a new kind of genetic algorithm called 'Breeder genetic algorithm' (BGA). The BGA is based on an artificial selection. The different components of this algorithm are based on the genetic algorithm. The performance of BGA is calculated on the execution of different multimodal functions. The authors present their implementation of mutation, crossover and the different selection function. In most of the test functions presented in this work, the search time can be represented by a function such as: n * ln(n) where n is the number of parameters. BGA is already applied to solve different problems in actual applications. The largest application is in the field of facial recognition where there are around 560 different parameters and the BGA can solve this problem easily.

The author of [19] uses genetic algorithms in order to solve complex problems in the field of chemistry. Indeed, some complex experiments need to define some parameters to be efficiency because the search space can be too large. The author defines a complex system like four different causes such as: high number of independent variables, very complex or irregular response surface, presence of discontinuities in the experimental domain and response to be optimised function of several 'subresponses'. In this article, the author has developed a genetic algorithm to find the most optimal parameters for a complex chemistry experiment. The benefit of this kind of algorithm according to the author is that genetic algorithms allow finding a good result quickly. Moreover, genetic algorithms can find a good result even when a maximum or a minimum cannot be detected. In this article, the developed genetic algorithm finds the best temperature in order to realize an experiment in the best conditions.

[4] presents different applications of evolutionary algorithm in different fields. The different advantages and disadvantages of multi objective evolutionary algorithms are discussed. The evolutionary algorithms are divided into two different groups. The first group is the non-elitist multi-objective evolutionary algorithms (MOEA) where the best results are not kept between different generations. The second group is elitist multi-objective evolutionary algorithms where the best results are kept between different generations. Genetic algorithms can be found within both groups, depending on how the genetic algorithm is defined. These algorithms are used to find patterns in a financial time series, to forecast the stock prices, to make data mining and other kinds of applications described in this paper. This article presents the evolutionary algorithms as a solution to make one task with more efficiency and accuracy.

In the last presented article [2], the author creates a radial basis function neural network (RBFNNs) to predict the output of a given model inputs. There are some parameters that need to be chosen carefully in order to obtain a good forecasting. The author uses a genetic algorithm (GA) to optimize these parameters. He writes that the initialisation in RBFNNs is actually important in order to make a good prediction. The created algorithm by the combination of RBFNNs and GA allow to obtain a better forecasting compared to the execution of RBFNNs without GA. Indeed, the normalized

root mean squared error (NRMSE) is 0.22 for the RBFNNs and GA and the execution time is 178 seconds; while NRMSE for the RBFNNs without GA is 0.27 and the execution time is 192 seconds. The performance and the computation time are improved when using GA.

Evolutionary algorithms can also be used to solve complex mathematics problems such as travelling salesperson problem [38]. This problem is hard to solve without the use of evolutionary algorithm when the number of city is large.

2.3 Energy forecasting

This section presents different articles that discuss the forecasting of energy data.

The article: 'A novel Spark-based multi-step forecasting algorithm for big data time series' [12] presents different kinds of techniques in order to forecast the consumption of electricity. The techniques are decision trees, two tree-based ensemble techniques (Gradient-Boosted and Random Forest) and a linear regression method. There are two different parameters that are taken into account to do the forecasting: the past values named W and the prediction problems named H. The data used are the consumption of electricity in Spain. This study compares the execution time and the mean relative error between the four different techniques. The random forest is the most efficient about the error rate (2.1863%) and the decision tree has the fastest execution (72 seconds).

Recently, the authors of [8] try to develop an algorithm to make the forecasting of electricity consumption of buildings to help the manager to reduce the consumption. The algorithm combines stacked autoencoders (SAE) with an extreme machine learning (ELM) approach. The SAE allows to obtain the energy building consumption and the ELM is used to forecast the future consumption of energy. The obtained results with this technique are compared with other techniques such as: backward propagation neural network (BPNN), support vector regression (SVR), generalized radial basis function neural network (GRBFNN) and multiple linear regression (MLR). The different results show that the technique developed in this paper is more efficient to predict the energy consumption of building compared to the other methods (BPNN, SVR, GRBFNN and MLR).

Research [27] of Ghulam Mohi Ud Din uses the electricity dataset collected from ISO New England for the period 2007 to 2012 around 52.600 measures to make a short-term forecasting. There are two kinds of deep learning algorithms that are used in this article: Feed-forward Deep Neural Networks (FFDNN) and Recurrent Deep Neural Networks (R-DNN). Different parameters are taken into account to do the forecasting such as: working and non-working days effects, time effect, temperature effects and other parameters. The result of this study shows that weather, time, holidays, lagged load and data distribution over the consumption period are parameters that influence the electricity consumption.

Another work is about deep learning algorithms for big data time series forecasting [41]. Indeed, this article introduces the use of this kind of algorithms to forecast Spain's

electricity consumption. This algorithm is based on the machine learning paradigm because the authors use a deep feed forward neural network. Exhaustive search is used in order to find the most optimal parameters of the deep learning algorithms. The authors compare this technique with other methods to predict the time series (linear regression, decision tree and other techniques) on the basis of the efficiency and the performance of the execution. The results show that the deep learning algorithm is the most efficient to forecast the consumption and the decision tree is the fastest to do the forecasting.

Our work represents the continuity of [41]. Indeed, the same data are used for both works as well as the basis of the used deep learning algorithm. But the difference in our work is the use of a genetic algorithm to optimize the parameters of the deep learning algorithm.

There is growing interest for energy forecasting because the energetic demand is increasing. So, the necessity to manage the energy consumption is emerging. Therefore, other techniques can be found to predict the energy data [39, 42].

Chapter 3

Deep Learning algorithms

In this chapter, the concept and the functioning of deep learning algorithms are introduced. In the second part, the different parameters to optimize are explained and their different possible values are mentioned. The execution time of deep learning is also discussed in this chapter.

3.1 Deep learning algorithms process

The figure below shows a schematically neural network as it is generated by the execution of deep learning algorithms.

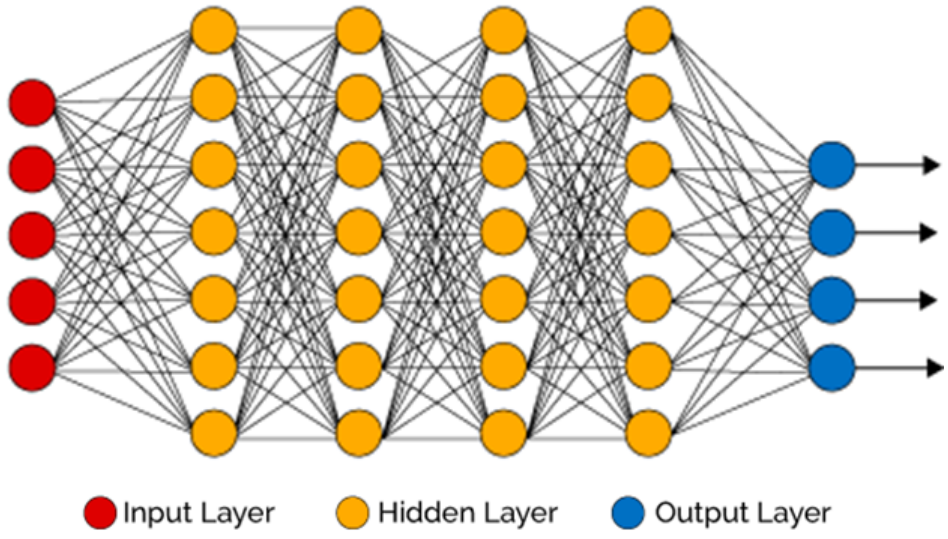

Fig. 3.1 : The process of deep learning algorithms [13]

The first layer is the input layer. The next layers are the hidden layers. The last layer is the output layer. All of these layers are composed with neurons and synapses.

3.1.1 Input layer

The input layer is the first layer of the neural network. This layer is composed of a number of nodes equal to the number of inputs. This layer allows to duplicate the data for each node also called neurons of the next layer. There are no operations at this stage so they are passive nodes. The neural network is called a fully interconnected structure because each neuron on one layer is connected with each neuron on the next layer. The data are sent through synapses that allow the connection between neurons of different layers [15].

In this study, there is only one kind of input which is a time series about the consumption of electricity in Spain. In addition, in our case, the number of neurons of the input layer is equal to the historical window size (for more information, cfr section 5.2).

3.1.2 Hidden layer

The hidden layers are the layers between the input and the output layers. Deep learning algorithms can have any number of hidden layers, and any number of neurons per hidden layer. The number of neurons per hidden layer and the number of hidden layers are usually defined by the user when the neural network is created.

3.1.2.1 Neurons

Neurons are the elements that allow to calculate the result. At this stage, the activation function can be applied to compute the sum of the received signals. The activation function is important in a neural network because this function decides whether a neuron is activated or not. A neuron is activated when the calculated result is higher than a threshold defined by an activation function. On the figure below, there is one value +1 that represents the bias term. This value is a neuron without connection with the previous layer. The bias term is always activated because it enables the regularization of the neural network's flexibility [6].

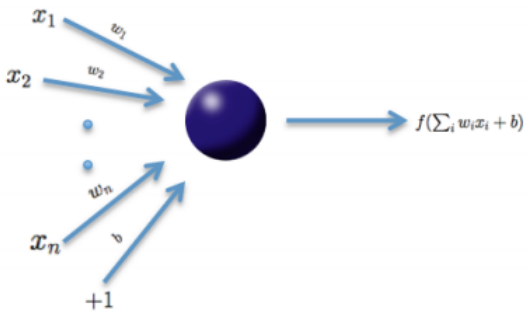

Fig. 3.2 : The representation of one neuron [7]

In figure 3.2, x_i represents the neuron i on the previous layer, w_i represents the signal that is sent to the neuron by the synapse i, the function f represents the nonlinear activation function and b represents the bias term for the neuron's activation threshold.

3.1.2.2 Synapses

Synapses connect neurons that are in different layers of deep learning algorithms. When the synapses receive information from the neurons, this information is multiplied by a synaptic weight that is associated to this synapse. The synaptic weight is defined during the training phase of the neural network (cfr section 3.1.4) [6].

$$w_i = x_i * s_i \tag{3.1}$$

x_i represents the signal sent by a neuron i on the previous layer and s_i represents synaptic weight for the synapse i.

3.1.3 Output layer

This is the last layer of deep learning algorithms. When the propagated information arrives at this layer, the execution is finished and the final calculated result by the deep learning algorithm can be retrieved.

In the case of our study, there is only one output unit because the deep learning algorithm calculates only the consumption of electricity of Spain for one hour.

3.1.4 Learning process

A deep learning algorithm has a learning process in order to modify the different parameters (weights of synapses and thresholds for the activation of neurons) in order to

produce a better output [6]. At the end of this process, the weights of synapses and the thresholds for the neurons activation of the neural network are the most efficient.

It is necessary to give data to the deep learning algorithm to do the learning phase. Data are generally divided into three different parts: the first part (training data) is used to train the neural network, the second part is used to validate the neural network (validation data) and the last part is used to test the neural network.

In our work, the data are divided into two different parts: 70% are used for the training of the neural network and 30% are used for the testing of the neural network. In the training data, 30% are also used to make the validation of the network.

In some case, the neural network is adapted only to respond to the training data. This is called the over-learning also known as overfitting. The opposite of overfitting is underfitting, it occurs when the neuronal network is not enough trained. The validation data are used to prevent overfitting. Indeed, the learning phase is finished when the generated error on the validation data by the neural network increases and not decreases [6]. After validation, the testing data are used to assess the accuracy of the neural network.

3.2 Parameters to optimize

The problem with deep learning algorithms is that some parameters need to be configured. The package H2O is used to create the deep learning algorithm [15, 7]. This package needs 8 different parameters to work correctly: number of layers, neurons per hidden layers, L1, rho, epsilon, activation function, distribution function and end metric. In the remainder of this paper, the L1 is denoted by lambda.

There are a lot of possible values for the different functions in deep learning algorithms (activation and distribution). The developed functions in the following sections are the functions that are implemented in the H2O package.

In this section, each parameter is explained (usefulness and interest) and the different possible values that it can take are expressed.

3.2.1 Number of hidden layers

This parameter defines the number of hidden layers in the deep learning algorithm.

The number of hidden layers depends on the complexity of the problem that the neural network must solve. Moreover, in [37], the authors said that the researchers have proven that having a large number of layers in a neural network does not improve a lot the obtained result.

Indeed, in [20], the authors showed that good results can be already obtained with

a neural network composed with 4 hidden layers. This neural network was applied on a problem of medium complexity. Moreover, if the number of hidden layers is too large, the execution time is larger.

In our work, the number of hidden layers is between 1 and 100.

3.2.2 Number of neurons per hidden layer

This parameter allows to define the number of neurons per hidden layer in the deep learning algorithm.

It is necessary to be careful when the number of neurons per hidden layer is chosen. Indeed, the number of neurons per hidden layer depends on the complexity of the problem. If there are too many neurons compared to the complexity of the problem to be solved, this can lead to overfitting. If there are not enough neurons compared to the complexity of the problem, this can lead to underfitting [32]. Both over and underfitting cases prevent a good functioning of the neural network.

According to the rule of thumb [32], the number of neurons per hidden layer is:

1. between the number of neurons in the input layer and in the output layer;
2. 2/3 of the number of neurons in the input layer plus the number of neurons in the output layer;
3. less than twice the number of neurons in the input layer.

In our study, it is not possible to follow this rule because the number of neurons per hidden layer would be too low. Indeed, for a historical window of 168, the number of neurons per hidden layer is between 1 and 335.

So, for our work, we have chosen to fix the number of neurons per hidden layer between 10 and 1000. Note that the number of neurons per hidden layer strongly influences the execution time (cfr section 3.3).

3.2.3 Value of lambda

Lambda is the parameter that will control the regularization of the model. Model regularization means the insertion of penalties in the model creation process in order to adjust as much as possible the values predicted with the actual values. The penalty is the sum of the absolute values of the weights received by the neuron [25]:

$$E = \lambda * \sum_{i=0}^{n} |w_i| \tag{3.2}$$

E represents the penalty that is applied, n is the number of weights received by the neurons, w represents the weight for the neuron i.

Lambda can take a value between 0 and 1. If the value is 0, the adjustment is ignored. If the value is close to 0, the regularization applied is smaller because the value of lambda is small.

In this study, the regularization does not need to be huge. For this reason, the value of lambda is between 0 and 1^{-9}. Furthermore, in [41], it is shown that the deep learning algorithm to forecast the consumption of electricity is most effective with a value of lambda close to 0.

3.2.4 Value of rho

Rho allows to manage the updating of different weight of synapses (cfr section 3.1.2.2) with previous results.

During the training of the neural network, the different weights are adjusted. Rho is used to maintain some consistency between the different updates of previous weights. It is better to modify slowly the weight between the updates because if it is too fast, it can lead to instabilities [15]. The formulas below show how the weights of synapses are updated during the training phase:

$$v_{t+1} = \rho * v_t - \alpha \nabla L(\theta_t) \tag{3.3}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{3.4}$$

v is a velocity vector used to modify the weight, θ represents a matrix of the weights, ρ is the coefficient of weight update and α is the learning rate [7].

The rho value is included between 0 and 1. If the value is close to 1, a strong consistency is kept between the weight updates. Otherwise, if the value is close to 0, there is not cohesiveness between the updates.

In our case, the rho value is between 0.99 and 1. This interval is chosen because it is recommended by the following document [7]. This is the reference document for the H2O package in which the best values are specified for each parameter of the neural network.

3.2.5 Value of epsilon

Epsilon prevents the deep learning algorithm from being stuck in local optimums or to skip a global optimum.

Indeed, in some cases, the neural network can find a good result and each following produced model is very sensitive to the precedent or the produced model can skip an optimum because the modification of the other parameters is always too important [7, 15].

The value of epsilon is between 0 and 1. This value allows to divide the learning rate by half. The chosen value determines the number of samples that are needed to divide by half the learning rate.

In our case, the epsilon value is between 0 and 1^{-9}. This interval is chosen because it is recommended by the following document [7]. This is the reference document for the H2O package in which the best values are specified for each parameter of the neural network.

3.2.6 Activation function

This parameter allows to define the activation function that is executed on the different signals entering a neuron. It allows to calculate a new signal to send to the nodes on the next layer.

In our study, the activation function can take 3 different values:

1. Tanh is the hyperbolic tangent defined by the following formula [7]:

$$f(\alpha) = \frac{e^{\alpha} - e^{-\alpha}}{e^{\alpha} + e^{-\alpha}} \, with f(.) \in [-1,1] \tag{3.5}$$

2. Rectifier is, also known as ramp function, defined by the following formula [7]:

$$f(\alpha) = max(0,\alpha) \, with f(.) \in R_{+} \tag{3.6}$$

3. Maxout is an activation function defined by the following formula [7]:

$$f(\alpha_1, \alpha_2) = max(\alpha_1, \alpha_2) \, with f(.) \in R \tag{3.7}$$

α is the weight that is calculated by the neuron with its different inputs (cfr Figure 3.2).

3.2.7 Distribution function

This parameter allows to define the distribution function that is used by the deep learning algorithm to determine if the different values that are calculated can be taken into account by the loss function. Furthermore, there is a link between the distribution that the user chooses and the loss function that the neural network selects.

When the distribution function is specified, the loss function is automatically selected by the neural network. The loss function is used to guide the training process of a neural network. The loss function calculates the difference between the produced and the expected value in order to guide the modifications of the different synaptic weights. It is an internal parameter of the model. For more information, we refer to [7, 15].

The distribution determines the different values that can be taken into account by the loss function. Indeed, in some case, one bad result can influence a lot the result of the loss function.

In our study, the distribution function can take 7 different values [35]: Gaussian, Poisson, Laplace, Tweedie, Huber, Gamma and Quantile.

3.2.8 End metric

This parameter allows to define the kind of metric that is used to stop early the training phase of the deep learning algorithm.

The end metric is linked with 2 other parameters: stopping_rounds and stopping_tolerance. The stopping_tolerance is the value by which a model must improve in order to continue the training phase. The default value for the tolerance is 0.001. The stopping_rounds is the number of rounds that the deep learning algorithm has in order to improve the chosen metric by the specified number defining by the stopping_tolerance. The default value for the number of rounds is 5 [15, 7].

For example, if the end metric is MSE, the tolerance is 0.001, the number of rounds is 5. It means that the deep learning will stop the training after reaching 5 rounds in a row in which the MSE value is not improved by 0.001.

In our case, the end metric can take 7 different values:

1. MSE is the mean square error value defined by the following formula [15]:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (a_i - p_i)^2 \tag{3.8}$$

   a represents the current value for index i in the training data, p represents the predicted value for index i, n represents the total number of values;

2. Deviance is the difference between an expected value and an observed value defined by the following formula [15]:

$$Deviance(a,p) = \sum_{i=1}^{n} (a_i, p_i) \tag{3.9}$$

   a represents the current value for index i in the training data, p represents the predicted value for index i, n represents the total number of values;

3. RMSE is the root mean squared error value defined by the following formula [15]:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (p_i - a_i)^2}{n}} \tag{3.10}$$

   a represents the current value for index i in the training data, p represents the predicted value for index i, n represents the total number of values;

4. MAE is the mean absolute error defined by the following formula [15]:

$$MAE = \frac{\sum_{i=1}^{n} |p_i - a_i|}{n} \tag{3.11}$$

a represents the current value for index i in the training data, p represents the predicted value for index i, n represents the total number of values.

5. RMSLE is the root mean squared log error defined by the following formula [15]:

$$RMSLE = \sqrt{\frac{\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2}{n}} \tag{3.12}$$

a represents the current value for index i in the training data, p represents the predicted value for index i, n represents the total number of values;

6. Mean_per_class_error is the mean per class error. This metric allows to determine the mean error for each different class. If there are different groups of data, it calculates the error individually [7];

7. Lift_top_group is a measure of the relative performance. The data are divided by groups, the default size is 20. For each group, the lift is calculated as the proportion of observations that are events in the group compared to the overall proportion of events [15].

3.3 Execution time of deep learning

This section shows how the number of neurons and the number of hidden layers can influence the execution time. The execution time is divided into two parts: the time to generate and to train the neural network and the time to forecast the consumption of electricity in Spain. The results shown in this part are the results of the experimentation led in the case of this study.

The other parameters of the deep learning algorithm are fixed as in the following. The value of Lambda is 1^{-9}. The value of Rho is 0.99. The value of Epsilon is 0. The activation function is Tanh. The distribution function is Gaussian and the end metric is MSE.

The table below shows the influence of the number of hidden layers and the number of neurons on the time needed to generate and to train the neural network but also to forecast the time series, the time is expressed in seconds for one subproblem :

neurons \ hidden layers	2	5	10	50	100
10	7.36	5.44	5.96	6.29	6.88
50	21.31	18.85	22.58	18.84	19.05
100	35.75	43.91	42.9	36.86	34.92
500	132.09	135.08	138.81	125.1	121.22
1000	257.74	251.05	250.18	253.23	249.10

Table 3.1: Execution time of deep learning algorithm (in seconds)

The mean time of the different results that are obtained by the execution of the deep learning algorithm is 89.42 seconds.

For instance, the deep learning algorithm with 100 neurons and 10 hidden layers takes 42.9 seconds. This time includes the time to generate and to train the neural network and to forecast the consumption of electricity in Spain.

The results show that the number of hidden layers has no influence on the execution time. The assumption can be made that the number of hidden layers is not large enough. It is the number of neurons per hidden layers that influences the execution time of the deep learning algorithm. Number of neurons influences the execution time because neurons are the processes where calculations are performed. Indeed, the time taken by the deep learning for 1000 neurons is around 35 times bigger than for 10 neurons with 2 hidden layers.

3.4 Parameter summary

The possible values for each numeric parameter of the deep learning algorithm are summarized in the next table:

Parameter	Minimum	Maximum
Number of hidden layer	2	100
Number of neurons per hidden layer	10	1000
lambda	0	1^{-9}
rho	0.99	1
epsilon	0	1^{-9}

Table 3.2: Different possible value for the numeric parameters

Chapter 4

Genetic algorithms

In this chapter, genetic algorithms are explained in detail. Each step in the operation of this type of algorithm is detailed. Furthermore, the parametrization of a genetic algorithm is explained and the different possible values for the parameters are listed. The package GA [38] is used to implement the genetic algorithm. When there are a lot of different choices (selection function, mutation and crossover), the developed ones are the different possibilities offered by the packge GA.

4.1 The genetic algorithms' process

The figure below shows the process flow of genetic algorithms. There are five different steps : initialisation, parent selection, application of the genetic operators (mutation and crossover), survivor selection and termination.

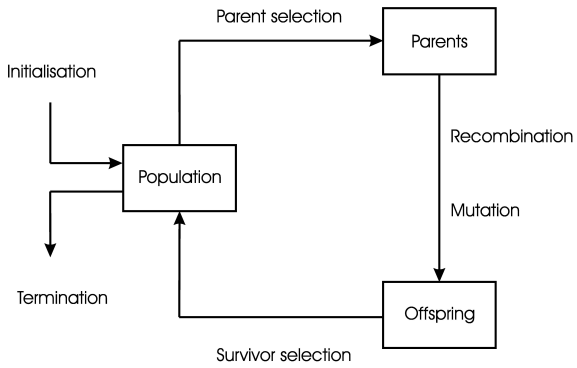

Fig. 4.1 : The process of genetic algorithms [1]

The explanations below of the different steps are based on [1].

4.1.1 Initialisation

The first step in genetic algorithms is to create the initial population. A population is a set of values that are possible solutions to the problem. In our case, the problem is to find the best set of parameters for the deep learning algorithm. So the population is a set that contains sets of potential values for the parameters.

This population is also called 'first population'. Indeed, it is the first one that is evaluated by the fitness function (cfr section 4.1.2). There are different ways to create the initial population, it can be generated randomly or it can be generated on the basis of a matrix, a vector or another set of values that is given by the user.

At the beginning of the genetic algorithms execution, it is necessary to define a size for the population and a number of generations. During the evolution, the defined size is kept for each iteration. If the size of the population is large, the chance to find a good solution is bigger because the diversity is potentially huge. However, the execution takes more time so it is necessary to find a compromise.

4.1.2 Parent selection

The second step in genetic algorithms is to select the values that must be kept to continue the execution. A fitness function is used to evaluate each value in the population and a selection function is used to determine how the selection is made. At the start of the genetic algorithm, it is necessary to define a percentage representing how many members of the population must be kept.

The fitness function allows to assign a quality measure to a value in the population. In this study, the fitness function is the deep learning algorithm that allows to calculate the mean relative error (cfr section 5.2). This value represents the quality of an element in the population. If the result is close to zero, it shows that the chosen parameters are effective.

The selection function allows to determine a way to select the values kept to generate the next population. There are different possibilities [38] :

1. linear-rank selection allows to attribute a value to each member of the population based on the result of the fitness function. The value 1 is for the worst and the value N that is the population size is for the best. This order is called the ranking. The normalized fitness values are a transformation of the fitness value in order to select with more efficiency the good solutions. They are calculated as follows:

$$F^N_{i_{temp}} = 2 - SP + \frac{2 * (SP - 1) * (i_{temp} - 1)}{N - 1} \tag{4.1}$$

N is the population size, SP is the selection pressure that is between [1.0,2.0] and i_{temp} is the position of the element in the ranking;

2. nonlinear-rank selection has the same ranking than the linear rank with the attribution value between 1 and N. The difference is the calculation of the normalized fitness values. It is defined as follows:

$$F^N_{i_{temp}} = N * \frac{X^{i_{temp}-1}}{\sum_{j=1}^{N} X^{j-1}} \tag{4.2}$$

N is the population size, i_{temp} is the position of the element in the ranking and X is calculated by the following formula :

$$0 = \sum_{j=1}^{N} X^{j-1} - \frac{N}{SP} \tag{4.3}$$

N is the population size and SP is the selection pressure selected between [1.0, N - 2.0]. This ranking selection permits a higher selective pressures than the linear ranking method because the solution of the polynomial equation has a bigger result for the high efficient element compared to the less efficient;

3. proportional selection, also known as roulette wheel selection, is a selection schema where a probability is associated to each value in the population. The probability associated is expressed by the following formula:

$$p_i = \frac{f_i}{\sum_{j=0}^{N} f_i} f_i \tag{4.4}$$

i is the individual in the population, f is the fitness value and N is the population size;

4. tournament selection chooses X individuals from the population and takes the best individual from this group into the intermediate population and the process is repeated N times;

5. fitness proportional selection with fitness linear scaling is based on the same principles as the proportional method but with a fitness linear scaling;

6. fitness proportional selection with Goldberg's sigma truncation scaling is based on the same principles as the proportional method but with a Goldberg's sigma truncation.

More information can be found about these selection techniques in the article of Adam Lipowski1and about roulette selection and in the article of Tobias Blickle about selection schemes [5, 21, 24].

For each member of the population, the result of the fitness function is calculated. After the evaluation, the chosen selection function is applied to determine which members of the population must be kept to create the next generation.

4.1.3 Genetic operators

The third step consists in the creation of the new population when the best values of the parent population are known. To create a new population from the selected values, there are two different methods that are used together: mutation and crossover.

Mutation

A mutation is a random alteration of the characteristics of an individual. It means that the different parts that constitute an element can be altered. For example, the value of one bit can be changed from 0 to 1.

The rate of mutation is low to keep an efficiency in the evolution of the population and not to fall into a random search. Mutation is nevertheless important because it allows to create a diversity between the population and it is able to avoid premature convergence.

There are different kinds of mutations that are studied:

1. uniform mutation changes the value of the element by another random selected value taking into account defined upper and lower bounds for this element;
2. nonuniform mutation allows to reduce the percentage of mutation through the different generations. At the end, the probability of mutation is closed to 0;
3. random mutation around the solution allows to change one value of an element by another value. By comparing results of random mutation, best mutation is identified and will be repeated later.

It is possible to find more information about the mutation allowed by GA and other kinds of mutations in the article of Soni Nitasha [30].

Crossover

Crossovers allow to create new offspring from the parent generation. It means that more than one parent is selected in order to use their genetic material to create one or more offspring in the new population.

The rate of crossover is expressed by a probability between 0 and 1. If the probability of crossover is high, there are more crossovers between parent populations. If the probability of crossover is low, there are less crossovers between parent populations.

There are different possibilities to execute one crossover:

1. single-point crossover divides two parents in two different parts. Afterwards, each part of each parent is combined to create two new elements in the population;

2. uniform crossover sets 0 or 1 for each part in the two new elements. The zero means that the part comes from the first parent and the one means that the part comes from the second parent;

3. whole arithmetic crossover works by taking the weighted sum with the same alpha of two parental parts for each parent. The alpha determines a coefficient of distribution. The value of alpha is between 0 and 1. If the alpha is equal to 0.5, it is the same computation than the single-point crossover. If the alpha is equal to 0 and 1, there is no effect. The following formula defines the mutation that is applied:

$$child_1 = \alpha * X + (1 - \alpha) * Y \tag{4.5}$$

$$child_2 = \alpha * Y + (1 - \alpha) * X \tag{4.6}$$

   X and Y are the different parts from the parents;

4. local arithmetic crossover is the same as whole arithmetic crossover, except that the alpha is randomly selected for each part location;

5. blend crossover combines two parents to generate offspring by sampling a new value in a define range with the maximum and the minimum of the parents.

More information about the crossover can be found in the article of Felipe Teodoro and in the article of Stjepan Picek [40, 34].

4.1.4 Survivor selection

The fourth step is also called the replacement. As mentioned in Section 3.1.1, the population size is always the same for each iteration. When the offspring is created with the different genetic operators, it is important to determine which individuals will be allowed into the next generation.

Survivor selection is performed after the creation of the offspring. This selection is normally based on the quality. So, the choice is made according to the fitness value although the concept of age can also be used for the parents, in order to replace some parents that have been generated too long ago.

For instance, suppose the population size is 100 and say that 200 children are created. The genetic algorithm must than select a new population of 100 members chosen from among the parents and the children.

4.1.5 Termination

The termination is the fifth and last step of genetic algorithm. There are different ways to stop a genetic algorithm: when an optimal fitness level is met or when the number of generation defined by the user is reached.

In some cases, a problem can have an optimal level. It means that the solution can not any more be more improved. When this level is reached, it is not necessary to continue the execution of the algorithm. In such a case, the algorithm is stopped before the number of generations defined by the user is reached.

In Section 4.1.1, a number of generations is defined. The genetic algorithm is stopped when this number of generation has been produced.

4.2 Parameters of the genetic algorithm

In our work, the package GA [38] for the language R is used. It allows to create genetic algorithms with some choices at the level of functions of selection, mutations, crossovers and others. The choice of each parameter and the different possible values are explained in what follows.

For some parameters, an experiment was conducted during this study in order to determine which value is the most efficient. The different tables in the following sections show the results that are obtained by the execution of the genetic algorithm with the deep learning algorithm as a fitness function. In this section, the deep learning algorithm is configured as shown in [41].

4.2.1 Type of genetic algorithm

This parameter is used to determine the type of genetic algorithm. It means the type of data that are generated by the genetic algorithm in order to solve optimization problems. The kind of genetic algorithm can only take three different values: 'binary', 'real-valued' and 'permutation'.

1. Binary optimizes the binary parameters. Normally, it is the optimization that is used for integer parameters.
2. Real-valued optimizes the real parameters. It uses real values to solve problems.
3. permutation classifies the different data. It is generally used for image recognition.

In this case, the parameters can take different kind of values such as string, integer or real. The easy way to solve this problem is to choose real-valued as a type of genetic algorithm.

4.2.2 Population size

This parameter allows to define the population size. This size is kept between the different generations. It is necessary to choose a number big enough in order to have a significant

diversity in the population. If the size is large, the execution takes more time. Normally, a population size of a genetic algorithm is between 75 and 150 [36].

In the following table, the results with different population sizes are showed. The other parameters are the same in all cases. The results have been put in the following table:

population size	Values of MRE (in %)
50	1.54501
100	1.50478
150	1.48975

Table 4.1: Results with different population size

The best results are obtained with a population size of 100 and 150. The problem of these sizes is the quite high execution time. To reduce it, the value 50 is chosen.

4.2.3 Number of generations

This parameter is the number of generations that are going to be created by the genetic algorithms. A large number allows to generate more results and to obtain potentially better results.

It is better to have more generations than to have a large population because there are more possible mutations and crossovers. However, in this study, it was not possible to use so many generations because the execution would have been too long.

Two different possibilities were tested: 50 generations and 100 generations. The other parameters are the same in all cases. The results have been put in the following table:

number of generations	Values of MRE (in %)
50	1.54501
100	1.50977

Table 4.2: Results with different generations

This table shows that the obtained result with more generations is better. So, the number of generations is fixed to 100.

4.2.4 Limit of generations

This parameter allows to fix a limit of generations. If the result is not enhanced after a number of generations, the execution of a genetic algorithm is stopped. This method allows to win time if the best result is already obtained.

In general, this number is close to the half of the number of generations only if the number of generations is large. In the other case, this number can be the same as the number of generations. Indeed, it is necessary to be careful because this parameter can stop the algorithm execution while the best possible result has not been reached yet. In this study, this parameter is fixed to 50.

4.2.5 Percentage of crossover

This parameter defines the probability of crossover between different parents in order to create new individuals in the population. This probability can be large because it allows to create different elements with more diversity.

Different possibilities were tested with different values for the percentage of crossover. The other parameters are the same in all cases. The results have been put in the following table:

percentage of crossover	Values of MRE (in %)
0.1	1.63578
0.5	1.59754
0.8	1.54501
0.9	1.56788

Table 4.3: Results with different percentage of crossover

This probability is between 0 and 1. The most appropriate value is 0.8 which is the default value for this parameter.

4.2.6 Percentage of elitism

This parameter defines the percentage of the population that must be kept from our generation to the next. Indeed, the best results that are obtained with the evaluation of the fitness function are kept during the different generations. If the percentage is large, the chances to obtain a better result are reduced.

Different possibilities were tested with different values of elitism. The other parameters are the same in all cases. The results have been put in the following table:

percentage of elitism	Values of MRE (in %)
0.05	1.54501
0.2	1.59712
0.6	1.65897
0.8	1.72478

Table 4.4: Results with different percentage of elitism

Results show that it is necessary to keep the best elements of the population to have a higher chance to obtain better elements thanks to mutations and crossovers. The chosen value is therefore 0.05 which is the default value for this parameter.

4.2.7 Percentage of mutation

This parameter defines the rate of mutations in the parent population. The rate of mutation is low to keep an efficiency in the evolution of the population and not to fall into a random search (cfr section 4.1.3).

Different possibilities were tested with different values. The other parameters are the same in all cases. The results have been put in the following table:

percentage of mutation	Values of MRE (in %)
0.1	1.54501
0.3	1.61578
0.5	1.7792
0.8	1.8579

Table 4.5: Results with different percentage of mutation

The default value is 0.1. This value is used because it is a low value for the percentage of mutations and it allows to obtain the best result.

4.2.8 Local optimization

This parameter is used to enable the optimization of results. This consists in modifying the generated population in order to try to reach a local optimum. It means modifying the different values to improve the obtained result from the initially generated population.

The value of this parameter is false because the optimization of the results leads to a greater execution time.

4.2.9 Selection function

There are 6 different possibilities of selection functions for real values : linear-rank selection, nonlinear-rank selection, proportional selection, tournament selection, fitness proportional selection with fitness linear scaling and fitness proportional selection with Goldberg's sigma truncation scaling (for explanation of each selection function cfr section 4.1.2).

The different possibilities were tested. The other parameters are the same in all cases. The results have been put in the following table:

function selection	Values of MRE (in %)
linear-rank selection	1.681
nonlinear-rank selection	1.8791
proportional selection	/
tournament selection	1.54501
fitness proportional selection with fitness linear scaling	/
fitness proportional selection with Goldberg's sigma truncation scaling	/

Table 4.6: Results with different selection functions

The different functions about proportional selection cannot be used because the fitness value is negative. This kind of function cannot work with negative value.

The best result is obtained with the tournament selection so it is this function that is used by the genetic algorithm.

4.2.10 Population function

This parameter is used to define a generation function for the initial population. The initial population is composed only of real values. The default function to generate real values is used.

4.2.11 Crossover function

The package GA defines different crossover functions such as : single-point crossover, uniform crossover, whole arithmetic crossover, local arithmetic crossover and blend crossover (for explanation of each crossover function cfr section 4.1.3).

The different possibilities were tested. The other parameters are the same in all cases. The results have been put in the following table:

crossover selection	Values of MRE (in %)
single-point crossover	1.6247
uniform crossover	1.59218
whole arithmetic crossover	1.55479
local arithmetic crossover	1.6247
blend crossover	1.54501

Table 4.7: Results with different crossover functions

The best result is obtained with the blend crossover. It is this function that is used for the genetic algorithm in this case.

4.2.12 Mutation function

The package GA allows to use different mutation functions such as : uniform random mutation, nonuniform random mutation and random mutation around the solution (for explanation of each selection function cfr section 4.1.3).

The different possibilities were tested. The other parameters are the same in all cases. The results have been put in the following table:

mutation selection	Values of MRE (in %)
uniform random mutation	1.57158
nonuniform random mutation	1.60477
random mutation around the solution	1.54501

Table 4.8: Results with different mutation functions

The best result is obtained with the random mutation around the solution. It is this function that is used for the genetic algorithm in this case.

4.2.13 Parameter summary

The values for each parameter of the genetic algorithm are summarized in the following table:

Parameters	Values
type	real_valued
population size	50
number of generations	100
limit of generations	50
percentage of crossover	0.8%
percentage of elitism	0.05%
percentage of mutation	0.1%
local optimization	FALSE
selection function	tournament selection
population function	gareal_Population
crossover function	blend crossover
mutation function	random mutation around the solution

Table 4.9: Parameters used for the genetic algorithm

The parameters listed in the table above obtained the best results. The parameter of population size is not the best because it is better to have more generations than a large population. There are more mutations and crossovers if there are more generations so the possibilities to have a large diversity is bigger.

In the section 3.3, the execution time for the deep learning is calculated. On average,

one execution of the deep learning algorithm takes around 89.42 seconds for a number of layers between 2 and 100 and for a number of neurons between 10 and 1000.

The execution of the genetic algorithm with the deep learning algorithm as a fitness function and with the parameters defined in the table 4.9 takes already around 5 days so if the population is doubled, the execution can take more than one week. It is necessary to enhance one of the parameters (population size or number of generations) but not both.

Chapter 5

Results

In this chapter, the data used for the experimentation are discussed and the obtained results are analysed. There are 3 kinds of results: results obtained with the deep learning, results obtained with a global optimization of the parameters (same parameters for the forecasted of next 24 hours) and finally results obtained with a specific optimization of the parameters (different parameters for the forecasted of next 24 hours).

The experimentation is executed only once for each following algorithm (deep learning without optimization, deep learning with a global optimization and deep learning with a specific optimization) because the execution of the deep learning combined with the genetic algorithm is time consuming.

At the end, the deep learning algorithm with and without parameters optimization using a genetic algorithm is compared with other techniques of forecasting.

5.1 Package's versions

In our work, different packages in R language are used to develop the deep learning algorithm and the genetic algorithm. However, some used packages require a specific version to work properly with the others.

The following table shows the different versions that are used for the different packages:

Package	Version
R	3.3.2
H2O	3.10.0.10
sparklyr	0.7.0 - 9035
rsparkling	0.1.1
dplyr	0.7.4

Table 5.1: The different versions of the used package

Sparklyr is a package in R language that allows to use spark for the language R. The spark language provides a way to manage big data as well as to perform some operations on it. The back-end is compatible with the package 'dplyr' and it provides an interface to Spark to build machine learning algorithms.

Rsparkling is a package in R language that is an extension to the 'sparklyr' package, it provides an interface for the language R for a cluster which is based on the H2O package.

Dplyr is a package in R language that provides a grammar for big data exploration and some operations that can be performed on it. It is a fast way to manage data like a frame in and out of the memory.

5.2 Used data

In this paper, time series data are used to carry out the different tasks and experiments. The concrete considered time series is related to the electricity consumption in Spain, expressed in megawatt (MW), from January 2007 to June 2016 which represents a large amount of data, around 500,000 stored measures available for processing and analysis. One measure is the consumption of electricity in Spain at a specific time over the period under consideration with a high sampling frequency (10 minutes).

The 500,000 measures are stored in a CSV file. These measures are divided into two different groups. The training set covers the period from January 1, 2007 at 00:00 to August 20, 2013 at 02:40 and the test set comprises the period from August 20, 2013 at 02:50 to June 21, 2016 at 23:40. This distribution of the data follows the rule of 70% for the training data and 30% for the testing data. In the 70% of the training data, 30% are also used to make the validation of the neural network.

The goal is the forecasting of electricity consumption for the next 24 hours based on the various readings that have been taken.

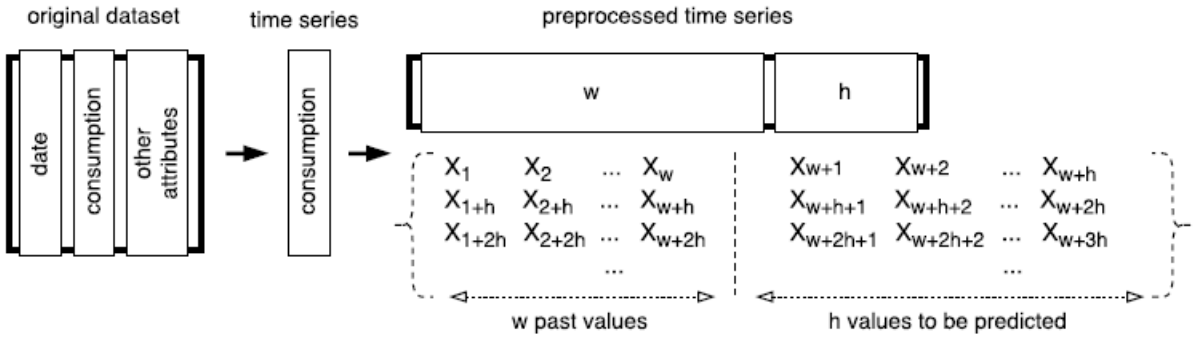

Fig. 5.1 : Explanation of the forecasting [41]

In the figure 6.1, the W represents the past values also called historical window. The historical window can be initialized to different values such as: 24, 48, 72, 96, 120, 144 and 168. These values of historical window correspond to 4, 8, 12, 16, 20, 24 and 28 past hours, respectively. The historical window represents the value that the user gives to the deep learning algorithm in order to forecast data. In [41], the author proves that the historical window is the most efficient when the window is equal to 168.

In the figure 6.1, the H represents the forecasted values. The forecasted values are the next 24 hours. In the remainder of this paper, the predicted hours are represented like subproblems. For example, subproblem 10 corresponds to the forecast for the tenth hour as specified in the introduction of this work.

5.3 Mean relative error

The data are divided into 2 different parts, 70% of the data are used for training and the last 30% are used for testing of the deep learning algorithm. The difference between the actual values corresponding to the testing data and the forecasted values is calculated with the mean relative error (MRE) expressed in percentage.

$$RME = 100 * \frac{1}{n} \sum_{i=1}^{n} \frac{|a_i - p_i|}{a_i} \tag{5.1}$$

a represents the actual value for index i in the testing data, p represents the forecasted value for index i, n represents the total number of values in the testing data [41].

The goal of the genetic algorithm is to find the best parameters for the deep learning in order to reduce the difference between predicted and actual values. If the MRE value is close to 0, the chosen parameters are efficient.

5.4 Analysis of the results

In this section, the different results obtained by the different algorithms are discussed and analysed. The first result corresponds to deep learning algorithm without optimization, the second one to deep learning algorithm with a global optimization of parameters and the last one to deep learning algorithm with a specific optimization of parameters. The historical window for these results is set to 168 (w = 168).

5.4.1 Results of deep learning without optimization

In this section, the used parameters and the obtained results without optimization of the parameters are summarized.

5.4.1.1 Parameters of the deep learning

The following table shows the different parameters that have been set for the execution of the deep learning algorithm:

Parameters	Values
Number of layer	5
Number of neurons per layer	100
lambda	0
Rho	0.99
Epsilon	1^{-9}
Activation function	Tanh
Distribution function	Gaussian
End metric	MSE

Table 5.2: Parameter used for the deep learning algorithm without optimization

This case is called the default case because it is based on deep learning with the default parameter of the algorithm as defined in [41]. These parameters are the basis parameters that are called also 'default parameters' in the remainder of this study.

5.4.1.2 Result of MRE for the 24 subproblems

The following table summarizes the results obtained by the execution of the deep learning algorithm without optimization of parameters:

Subproblem	MRE (in %)
1	0.977321
2	1.162216
3	1.181268
4	1.30299
5	1.458626
6	1.524349
7	1.492074
8	1.833509
9	1.864269
10	1.871364
11	1.791695
12	2.062735
13	2.404409
14	2.057679
15	2.1304
16	2.162826
17	2.238509
18	2.364257
19	2.303624
20	2.620436
21	2.551586
22	2.598092
23	2.588869
24	2.604855

Table 5.3: Values of MRE in % without optimization for the 24 subproblems

The mean of MRE for the 24 subproblems is 1.964498%. It is possible to forecast the consumption of electricity in Spain for one day with an error rate close to 2%.

For the first subproblem the MRE is under 1% and this is a good result because the prediction is almost correct. Furthermore, it is at the 12^{th} subproblem that the MRE rises above 2%. It can be explained because it is easier to predict the first value than the twelfth. Indeed, it is harder to make a good forecasting of data when the prediction horizon is large.

The obtained results by the deep learning algorithm without modification is rather good because the error rate is low, but the error increases when the prediction horizon is extended.

5.4.2 Results of deep learning with a global optimization

This section summarizes the parameters found and the results obtained with a global optimization of the parameters with the genetic algorithm. A global optimization means that the same parameters for the deep learning algorithm are used for every subproblem.

5.4.2.1 Parameters of the deep learning

The following table shows the different parameters that have been found for the execution of the deep learning algorithm. These parameters are found by the execution of the genetic algorithm:

Parameters	Values
Number of layer	58
Number of neurons per layer	749
lambda	0
Rho	0.9962764
Epsilon	0
Activation function	Maxout
Distribution function	huber
End metric	mae

Table 5.4: Parameters found for the deep learning algorithm with a global optimization

These parameters are used for every subproblem without modification. They change only when a new population is generated by the genetic algorithm.

5.4.2.2 Result of MRE for the 24 subproblems

The following table gives the results obtained by the execution of the deep learning algorithm with a global optimization of parameters:

Subproblem	MRE (in %)
1	0.57894
2	0.658781
3	0.89785
4	0.925789
5	1.124147
6	1.097782
7	1.212547
8	1.347214
9	1.427898
10	1.435147
11	1.45797
12	1.493659
13	1.517423
14	1.524831
15	1.621479
16	1.669751
17	1.698959
18	1.74681
19	1.835104
20	1.927412
21	2.024793
22	2.081479
23	2.109712
24	2.148793

Table 5.5: Values of MRE in % with a global optimization for the 24 subproblems

The mean of MRE for the 24 subproblems is 1.48184458%. It is possible to forecast the consumption of electricity in Spain for one day with an error rate close to 1.5%.

In this case, the MRE is under 1% until the 4^{th} subproblem. It is only at the 21^{st} subproblem that the result is over 2%.

The obtained results with a global optimization are good because it is possible to predict the consumption of electricity in Spain with an error rate under 1.5% for the first half of the subproblems.

5.4.3 Results of deep learning with a specific optimization

In this section, the found parameters and the obtained results are related to specific optimization of the parameters with the genetic algorithm. A specific optimization means searching the most efficient parameters for each subproblem. The parameters can change between each subproblem.

5.4.3.1 Parameters of the deep learning

The following table shows the different parameters that have been found for the execution of the deep learning algorithm. These parameters are found by the execution of the genetic algorithm:

Subproblem	layers	Neurons	Lambda	Rho	Epsilon	activation	distribution	End metric
1	52	942	4.090454^{-10}	0.9952963	6.434323^{-12}	Tanh	gaussian	deviance
2	68	921	0	0.9976888	0	Maxout	huber	MSE
3	75	880	0	0.9956808	0	Maxout	huber	deviance
4	68	921	0	0.9983268	0	Maxout	huber	MSE
5	88	504	0	0.9983331	0	Maxout	huber	deviance
6	80	789	0	0.9974687	0	Maxout	huber	MSE
7	74	892	0	0.9995477	0	Maxout	huber	rmsle
8	46	300	0	0.9971749	0	Maxout	huber	mae
9	75	889	5.570327^{-10}	0.9904759	6.743984^{-10}	Tanh	gaussian	mean_per_class_error
10	25	852	0	0.9977593	0	Maxout	huber	rmsle
11	58	843	3.691579^{-10}	0.9992602	2.451509^{-10}	Tanh	gaussian	rmse
12	41	491	0	0.9966297	0	Maxout	huber	rmsle
13	17	552	0	0.9930415	0	Maxout	huber	MSE
14	26	661	0	0.992265	0	Maxout	huber	mae
15	89	811	5.606931^{-10}	0.9929693	4.229728^{-10}	Tanh	gaussian	rmse
16	98	697	0	0.9978715	0	Maxout	huber	mae
17	74	478	1.459971^{-10}	0.9987115	3.584887^{-10}	Tanh	gaussian	deviance
18	62	705	2.740637^{-10}	0.9949344	6.635605^{-10}	Tanh	gaussian	mae
19	65	879	0	0.9907152	0	Maxout	huber	mae
20	81	780	7.620906^{-10}	0.9907152	5.207976^{-10}	Tanh	gaussian	MSE
21	27	931	0	0.997667	0	Maxout	huber	mae
22	95	745	0	0.9974681	0	Maxout	huber	deviance
23	41	923	0	0.9951057	0	Maxout	huber	MSE
24	80	754	0	0.9963058	0	Maxout	huber	mae

Table 5.6: Parameters found for the deep learning algorithm with a specific optimization

In the default case, the number of layer is 5. The optimization shows that the number of layers need to be larger than 50 for most of the subproblems. However, there are some subproblems where less than 50 layers seem to be optimal.

The number of neurons required is above 700 for the majority of the subproblems. There are some subproblems that have more than 900 neurons. This is in most cases more than the default case with only 100 neurons. The number of neurons needs to be huge to forecast with precision the consumption of electricity in Spain.

The value of lambda and epsilon are close to 0 or equal to 0; very similar to the default values for these parameters (0 and 1^{-9}). Furthermore, when the value of lambda is equal to 0, the value of rho is equal to 0 too.

Rho values are close to 1. The default value for this parameter is 0.99 and the value found by the evolutionary algorithm is bigger than the default.

The activation function is almost the same for the 24 different subproblems namely Maxout. In the other cases, the Tanh is chosen like in the default case. So, we can suppose that Maxout function is the most effective to forecast the data about the consumption of electricity in Spain.

It is the same for the distribution function. In most cases, it is the Huber function that is the more effective. In the other cases, it is the Gaussian function that is chosen.

There is one link between the activation and the distribution function. When the activation function is Maxout, the distribution function is always Huber, while the distribution function is always Gaussian when activation function is Tanh.

Furthermore, the assumption can be made that there is a link between these functions (activation and distribution) and the values of rho and lambda. Indeed, when the activation function is Maxout and the distribution function is Huber, the values of lambda and rho are equal to 0. And when the activation function is Tanh and the distribution function is Gaussian, the values of lambda and rho are close to 0 but not equal.

The last parameter is the end metric and there is no pattern for this one. There are 6 different values out of the 7 possibilities. The only one that is not used is the lift_top_group.

The specific optimization is interesting because it shows that the parameters can be different for each subproblem. Moreover, the parameters chosen in the default case are not the most effective to forecast data.

5.4.3.2 Result of MRE for the 24 subproblems

The following table resumes the obtained results by the execution of the deep learning algorithm with a global optimization of parameters:

Subproblem	MRE (in %)
1	0.55915
2	0.725149
3	0.859665
4	0.913307
5	1.017884
6	1.13408
7	1.288955
8	1.23177
9	1.414811
10	1.396267
11	1.435687
12	1.487978
13	1.508399
14	1.539579
15	1.510583
16	1.616487
17	1.71281
18	1.735445
19	1.857298
20	1.760722
21	2.015574
22	2.049844
23	2.066408
24	2.071107

Table 5.7: Values of MRE in % with a specific optimization for the 24 subproblems

The mean of MRE for the 24 subproblems is 1.45454%. It is possible to forecast the consumption of electricity in Spain for one day with an error rate close to 1.45%.

In this case, the MRE is under 1% until the 4^{th} subproblem. It is only at the 21^{st} subproblem that the result is over 2%. This observation is the same as the deep learning algorithm with a global optimization.

The obtained results with a specific optimization are good because it is possible to predict the consumption of electricity in Spain with an error rate under 1.5% for half of the subproblems.

The following graphic shows the evolution of the genetic algorithm for the subproblem 1 during the 100 generations:

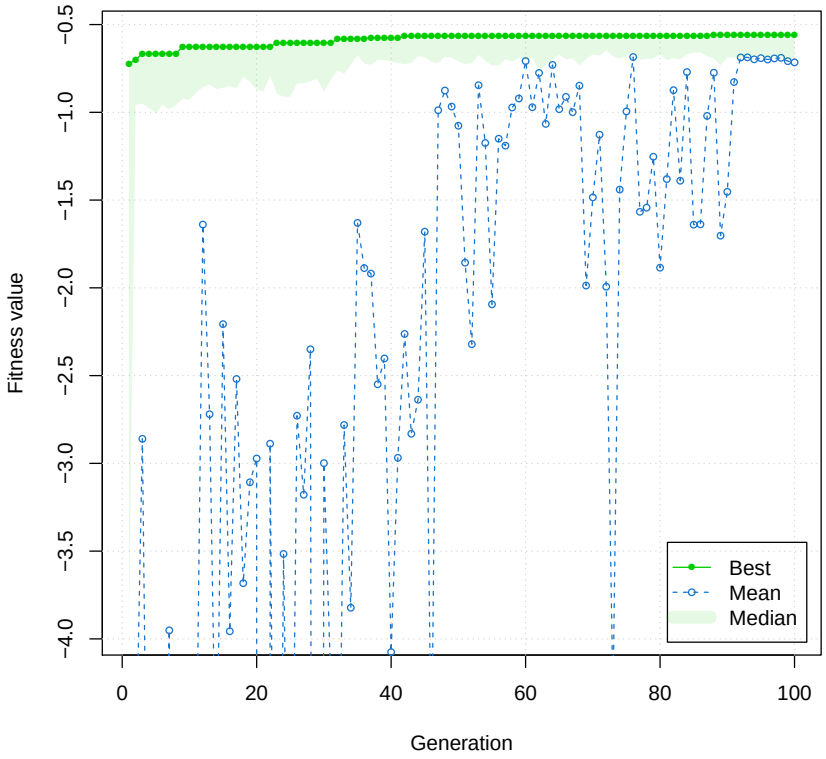

Fig. 5.2 : Evolution of genetic algorithm for the subproblem 1

The other charts for every subproblem are in the appendix A.

On this graph, the evolution of the best result is shown with the green curve. It starts around 0.7% to finish at 0.55915% (the absolute values are considered). During the 100 generations, the best result is enhanced 6 times. Most of these improvements are at the beginning because it is easier to enhance the result at the start than at the end.

The blue curve shows the mean of the MRE for all results of the same generation. The last color is the median of this graphic.

5.4.4 Comparison between the 3 techniques

The following table summarizes the results obtained by the execution of the deep learning algorithm with and without optimization :

Techniques	MRE (in %)
Deep learning without optimization	1.964498
Deep learning with a global optimization	1.48184458
Deep learning with a specific optimization	1.45454

Table 5.8: MRE in % for the 3 different techniques of deep learning

The best result is obtained by the deep learning with a specific optimization of the parameters. The worst is the deep learning without optimization. These results show that the deep learning needs to be parametrize carefully.

Indeed, the result of deep learning without optimization is 1.96% and the result of deep learning with a specific optimization is around 1.45% so the result is improved by 0.5% by choosing the right parameters. It is not a minor improvement. It can be explained because the deep learning is very sensitive to the initialization. The activation function, the distribution function and the other parameters influence a lot the result obtained by the execution of the deep learning algorithm. So, it is worthy to chose them carefully.

The results between the specific and the global parameters are really close. There is not a big difference. The deep learning with a specific optimization is chosen because this technique allows to obtain the best result.

5.5 Comparison with other techniques

In this section, the results obtained by the deep learning algorithm and by the deep learning algorithms with a specific optimization are compared with other techniques to forecast the data. Compared techniques are: linear regression (LR), decision tree (DT), gradient-boosted trees (GBT), random forest (RF), deep learning without optimization (DP) and deep learning with a specific optimization (DP + EA). More information about these techniques can be found in the following article [12]. In this part, the deep learning with a global optimization is not taken into account because the experiments of this technique were not executed for each historical window.

The following table summarizes the MRE results in % obtained by the execution of the different techniques to forecast the data for different historical windows:

W	LR	DT	GBT	RF	DP	DP + EA
168	10.0876	2.8958	2.7431	2.0831	1.964498	1.45454
144	7.3397	2.9271	2.7520	2.1863	2.180803	1.506478
120	10.4554	3.1801	3.0165	2.4160	2.487146	1.604587
96	13.5324	3.3032	3.1334	2.7045	2.635448	1.847836
72	14.0328	3.4386	3.2383	2.8912	2.835173	2.084307
48	13.9949	4.0322	3.7019	3.5969	2.993675	2.383614
24	10.8781	4.7625	4.4633	4.4846	4.003731	3.0036

Table 5.9: MRE in % for different forecasting techniques

Experimentation results of LR, DT, GBT, RF are drawn from the following research [12]. The nature of the techniques that are compared is different. The linear regression is a linear approach. The decision tree is a decision support. GBT and RF are two other tree approaches. DP and DP + EA are based on the deep learning approach.

The first observation is that in general the obtained results with a large historical window are better than with a smaller size. It can be explained because it is easier to forecast the data when more time is taken into account.

Secondly, the results obtained by the linear regression are the worst compared to all other techniques. Best results are obtained by DP + EA. It can be explained because the deep learning creates a neural network that is trained in order to predict with more efficiency.

Thirdly, the different results show a MRE improvement when the size of w grows. It can be explained because it is easier to forecast the data with more past values. Indeed, the best result is obtained when the historical window is 168 (w = 168) except for LR that is a better MRE with the historical window 144. For all tree-based methods and the deep learning methods, the improvement of MRE between the historical window 144 and 168 is not big.

Fourthly, results obtained by DP are better than results obtained by RF which is the best method between LR, GBT and DT. RF results are only better in the historical window 120. The difference for 144, 120, 96 and 72 is small but for other sizes the difference is bigger.

Fifthly, results obtained by DP + EA are better for every historical window. Moreover, DP + EA is just an optimization of DP so the obtained results are better than DP. However, it is interesting to observe that results obtained by all methods are really close. Only LR method produces systematically weaker results.

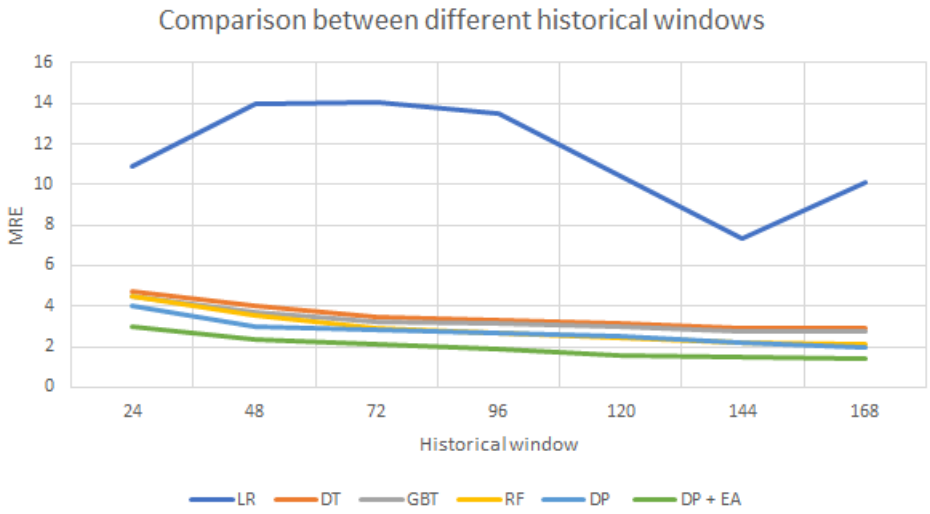

Fig. 5.3 : Evolution of the MRE in % according to the historical window for the different techniques

Figure 5.3 shows the data in the table 5.9 as a graph. This figure highlights that the results obtained by the different techniques are close excepted for the linear regression technique that has the worst results.

Finally, these results show that the deep learning is the most effective technique to forecast the consumption of electricity in Spain. Furthermore, the parameters for this technique are really important because they can influence a lot the results obtained by deep learning algorithms. Indeed, deep learning algorithm with a specific optimization of the parameters has better results than without optimization.

Chapter 6

Conclusion

The aim of this work is to use an evolutionary algorithm to efficiently parametrize a deep learning algorithm in order to forecast electricity consumption of Spain. The developed algorithm in this work is based on two principles: deep learning algorithms and genetic algorithms.

The deep learning algorithm allows to forecast the consumption of electricity through the creation of a neural network. The difficulty is the parametrization of this neural network because it is necessary to make some choices (e.g. activation function, distribution function and other choices) (cfr chapter 3).

The genetic algorithm is used to parametrize the different parameters of the neural network through an evolution principle. Indeed, the genetic algorithm allows to evaluate different possible solutions and to determine which are the most efficient in order to solve the problem (cfr chapter 4).

The developed algorithm in this study is compared to other methods to forecast data such as: linear regression (LR), decision tree (DT), gradient-boosted trees (GBT), random forest (RF) and deep learning without optimization (DP). The objective is to analyse if the developed algorithm is more efficient than the other methods that is to say it predicts the data with more accuracy.

The comparison between the different techniques shows that the deep learning algorithm is the most efficient way to forecast the consumption of electricity of Spain. Indeed, the mean relative error (MRE) is the weakest with this kind of algorithm (cfr Table 5.8).

Moreover, this study shows that the deep learning algorithm is really sensitive to the initialization. Indeed, the chosen parameters for the deep learning algorithm by the evolutionary algorithm decreases the MRE obtained by the deep learning algorithm by 0.5%. Compared to the MRE of 2% obtained by the default case, this is a significant improvement.

In addition, the developed algorithm, namely a deep learning algorithm combined with an evolutionary algorithm, is the most efficient method in order to forecast the

consumption of electricity in Spain. Indeed, the consumption of electricity for one day can be predicted with an accuracy of 1.45454%.

The research presented in our work can be improved. Indeed, it is possible to identify multiple future improvements to this work for example: managing different input types for the deep learning algorithm and relaxing the chosen constraints for some parameters of the deep learning algorithms.

For the moment, the only input of the deep learning algorithm is the different measures of the consumption of electricity in Spain (cfr section 5.2). It is possible to modify the deep learning algorithm in order to take into account other inputs such as weather, period of the year, bank holidays and other kinds of data. These different inputs can be useful to predict with more accuracy the consumption of electricity. For instance, the weather can be another parameter of the deep learning algorithm because if there is sun, people typically use less electricity or people can use more electricity in the summer for air-conditioning.

Some parameters of the deep learning algorithm had a constrained interval of values (number of neurons, number of layers, lambda, rho and epsilon). An interesting study would be to increase the interval of values for these parameters in order to see if the MRE obtained by the execution of the deep learning algorithm can still be improved.

Bibliography

[1] Eiben A.E. and Smith J.E. *Introduction to Evolutionary Computing*. Spring, 2003.

[2] Mohammed Awad. Optimization rbfnns parameters using genetic algorithms: Applied on function approximation. *International Journal of Computer Science and Security (IJCSS)*, 4:295–307, June 2010.

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, February 2012.

[4] Shifali Bhargava. A note on evolutionary algorithms and its applications. *Adults Learning Mathematics*, 8:31–45, April 2013.

[5] Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in genetic algorithms. *Evolutionary Computation*, 4:361–394, December 1996.

[6] Nikhil Buduma and Nicholas Locascio. *Fundamentals of deep learning*. O'Reilly Media, June 2017.

[7] Arno Candel and Erin Ledell. Deep learning with h2o. April 2018.

[8] Li Chengdong, Ding Zixiang, Zhao Dongbin, Yi Jianqiang, and Zhang Guiqing. Building energy consumption prediction: An extreme deep learning approach. *Energies*, pages 1–20, October 2017.

[9] Janssen Dale and Janssen Cory. techopedia. https://www.techopedia.com/, 2018.

[10] Kenneth De JongDavid, David B. Fogel, and Hans-Paul Schwefel. A history of evolutionary computation. *Handbook of Evolutionary Computation*, pages 3–17, January 1997.

[11] Lee Donghyun, Suna Kang, and Jungwoo Shin. Using deep learning techniques to forecast environmental consumption level. *Understanding User satisfaction Evaluation in low occupancy Sustainable Workplaces*, 9, October 2016.

[12] A. Galicia, J. F. , F. Martinez-Alvarez, and A. Troncoso. A novel spark-based multi-step forecasting algorithm for big data time series. August 2017.

[13] Anna Gomez. Deep learning in digital pathology. http://www.global-engage.com/life-science/deep-learning-in-digital-pathology/, February 2018.

[14] Varun Gulshan, Lily Peng, and Marc Coram. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama, Innovations in health care delivery*, December 2016.

[15] H2O.ai. H2o documentation. http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/deep-learning.html, 2016-2017.

[16] Jassim Hassanean S. H., Lu Weizhuo, and Thomas Olofsson. Predicting energy consumption and co2 emissions of excavators in earthwork operations: An artificial neural network model. *Fostering sustainable urban-rural linkages through local food supply : A transfnational analysis of collaborative food alliances*, 9, July 2017.

[17] Jack Kelly and William Knottenbelt. Neural nilm : Deep neural networks applied to energy disaggregation. *ACM International Conference on Embedded Systems For Energy-Efficient Built Environments*, November 2015.

[18] Justin Ker, Lipo Wang, Jai Rao, and Tchoyoson Lim. Deep learning applications in medical image analysis. In *IEEE Access*, pages 9375–9389. IEEE, December 2017.

[19] Riccardo Leardi. Genetic algorithms in chemistry. *Journal of Chromatography A*, 22:226–233, July 2007.

[20] Thomas Likewin, Kumar M. V. Manoj, and B. Annappa. Discovery of optimal neurons and hidden layers in feed-forward neural network. In *IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech)*, pages 286–291. IEEE, August 2016.

[21] Adam Lipowski1and and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391:2193–2196, March 2012.

[22] F. Martínez-Álvarez, A. Troncoso, G. Asencio-Cortés, and J. C. Riquelme. A survey on data mining techniques applied to energy time series forecasting. *Energies*, 8:13162–13193, November 2015.

[23] Antonino Marvuglia and Antonio Messineo. Using recurrent artificial neural networks to forecast household electricity consumption. *Energy procedia*, 14:45–55, 2011.

[24] Mohamed Marzouk and Osama Moselhi. Constraint-based genetic algorithm for earthmoving fleet selection. *Canadian Journal of Civil Engineering*, 30:673–683, 2003.

[25] James D. McCaffrey. Implementing neural network l1 regularization. https://jamesmccaffrey.wordpress.com/2017/06/27/implementing-neural-network-l1-regularization/, June 2017.

[26] Heinz Mühlenbein and Dirk Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm i. continuous parameter optimization. In *Evolutionary Computation*, volume 1, pages 25–49. IEEE, March 1993.

[27] Ghulam Mohi Ud Din and Angelos K. Marnerides. Short term power load forecasting using deep neural networks. In *Computing, Networking and Communications (ICNC), 2017 International Conference on*, pages 594–598. IEEE, January 2017.

[28] Michael Nielsen. Neural networks and deep learning. http://neuralnetworksanddeeplearning.com/index.html, December 2017.

[29] Jürg Nievergelt. Exhaustive search, combinatorial optimization and enumeration: Exploring the potential of raw computing power. *SOFSEM 2000: Theory and Practice of Informatics*, 1963:18–35, January 2002.

[30] Soni Nitasha and Tapas Dr Kumar. Study of various mutation operators in genetic algorithms. *International Journal of Computer Science and Information Technologies*, 5:4519–4521, 2014.

[31] A. Ortiz, J. Munilla, J.M. Gorriz, and J. Ramirez. Ensembles of deep learning architectures for the early diagnosis of the alzheimer's disease. *International Journal of Neural Systems*, 23, November 2016.

[32] Foram S. Panchall and Mahesh Panchal. Review on methods of selecting number of hidden nodes in artificial neural network. *International Journal of Computer Science and Mobile Computing*, page 455–464, November 2014.

[33] Daniel Peralta, Sara del Río, Sergio Ramírez-Gallego, Isaac Triguero, Jose M. Benitez, and Francisco Herrera1. Evolutionary feature selection for big data classification: A mapreduce approach. *Mathematical Problems in Engineering*, 2015, June 2015.

[34] Stjepan Picek, Domagoj Jakobovic, and Marin Golub. On the recombination operator in the real-coded genetic algorithms. In *IEEE Congress on Evolutionary Computation*, pages 3103–3110. IEEE, June 2013.

[35] Greg Ridgeway. Generalized boosted models: A guide to the gbm package. *researchgate*, April 2006.

[36] Olympia Roeva, Stefka Fidanova, and Marcin Paprzycki. Influence of the population size on the genetic algorithm performance in case of cultivation process modelling. pages 371–376. IEEE, September 2013.

[37] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, January 2015.

[38] Luca Scrucca. Ga: A package for genetic algorithms in r. *Journal of Statistical Software*, 53, April 2013.

[39] L. Suganthi and A. Samuel Anand. Energy models for demand forecasting — a review. *Renewable and Sustainable Energy Reviews*, 16:1223–1240, February 2012.

[40] Felipe Teodoro, A. M. Lima Clodoaldo, and Marques Peres Sarajane. Supply chain management and genetic algorithm: introducing a new hybrid genetic crossover operator. In *X Encontro Nacional de Inteligência Artificial e Computacional*, October 2013.

[41] J. F. Torres, A. Galicia, A. Troncoso, and F. Martínez-Álvarez. A novel scalable approach based on deep learning for big data time series forecasting. 2017.

[42] Jyothi Varanasi and M. M. Tripathi. A comparative study of wind power forecasting techniques — a review article. In *3rd International Conference on Computing for Sustainable Global Development)*, pages 3649–3655. IEEE, March 2016.

[43] Li Xiang, Peng Ling, Hu Yuan, Chi Tianhe, and Shao Jing. Deep learning architecture for air quality predictions. *Environmental Science and Pollution Research*, 23:22408–22417, November 2016.

Appendices

Appendix A

Evolution of the genetic algorithm for the subproblems

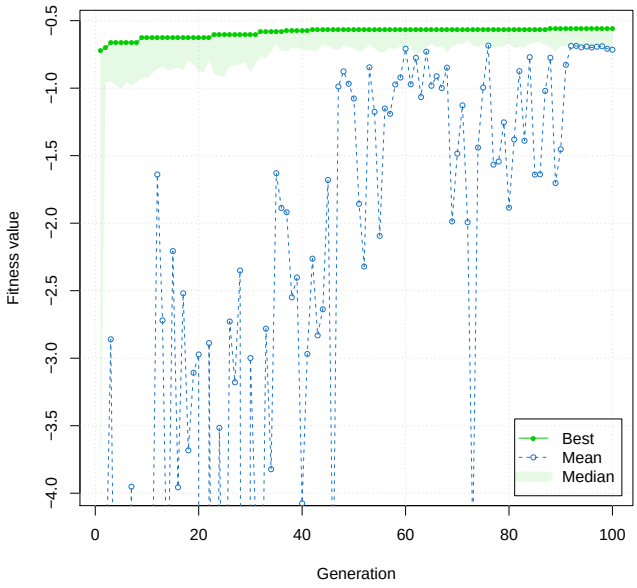

Fig. A.1 : Evolution of genetic algorithm for the subproblem 1

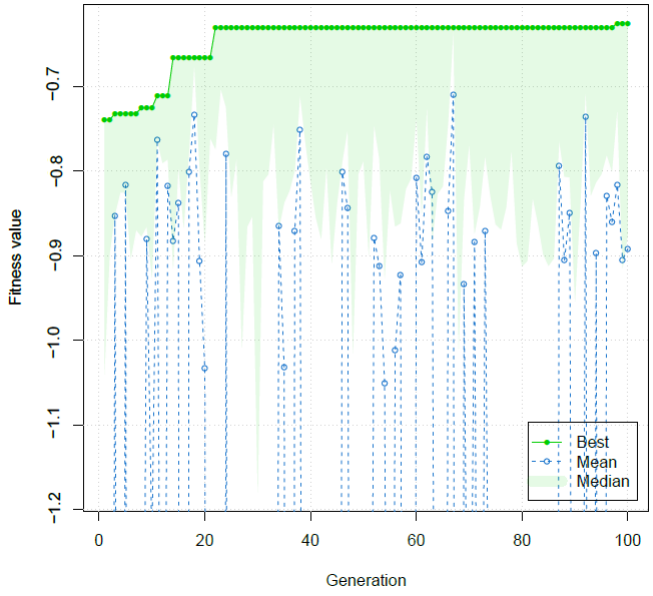

Fig. A.2 : Evolution of genetic algorithm for the subproblem 2

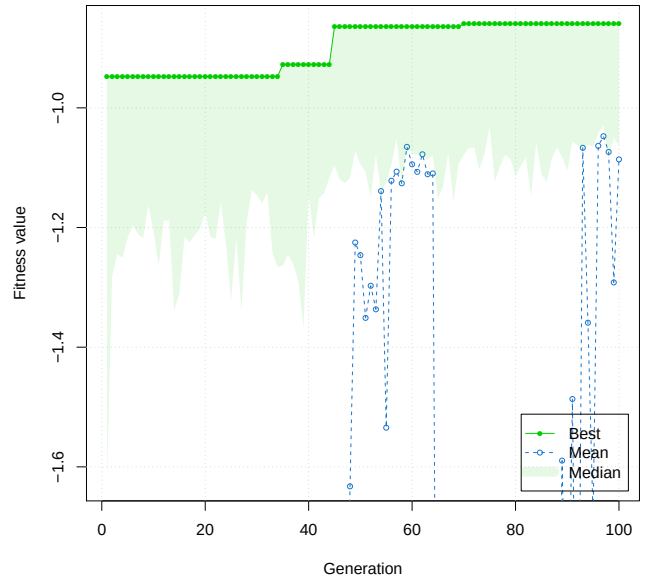

Fig. A.3 : Evolution of genetic algorithm for the subproblem 3

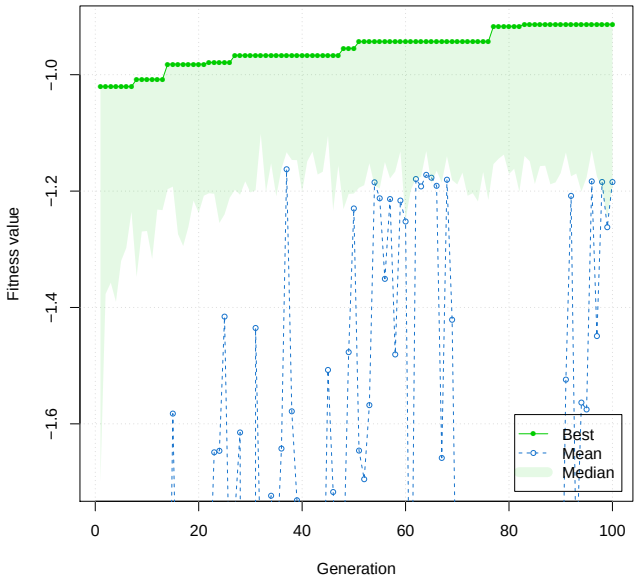

Fig. A.4 : Evolution of genetic algorithm for the subproblem 4

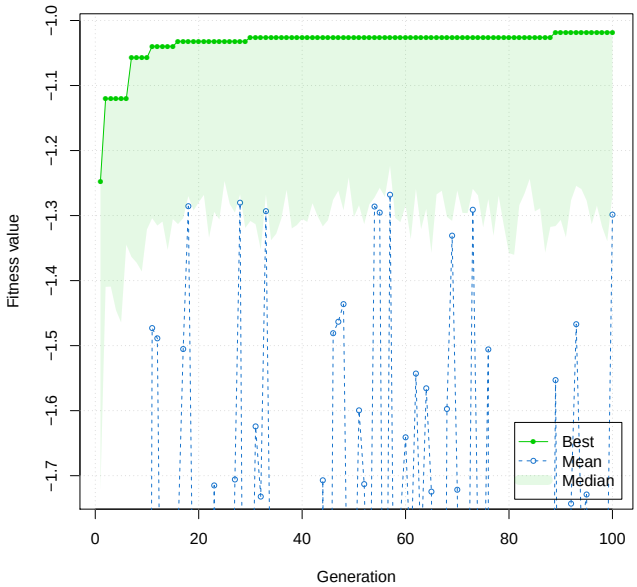

Fig. A.5 : Evolution of genetic algorithm for the subproblem 5

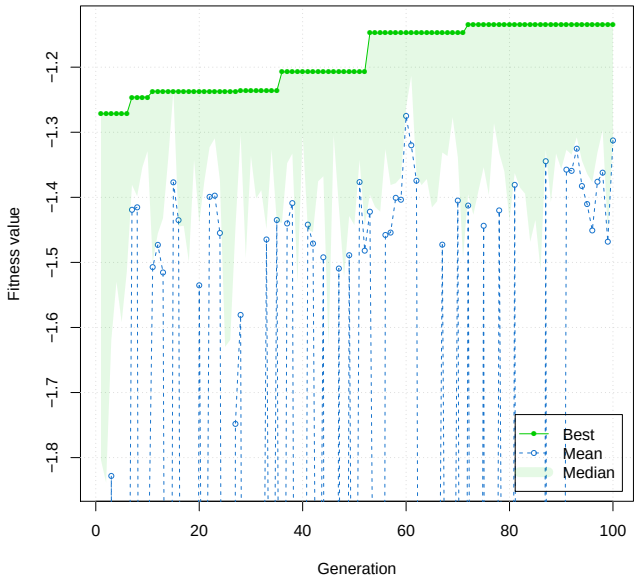

Fig. A.6 : Evolution of genetic algorithm for the subproblem 6

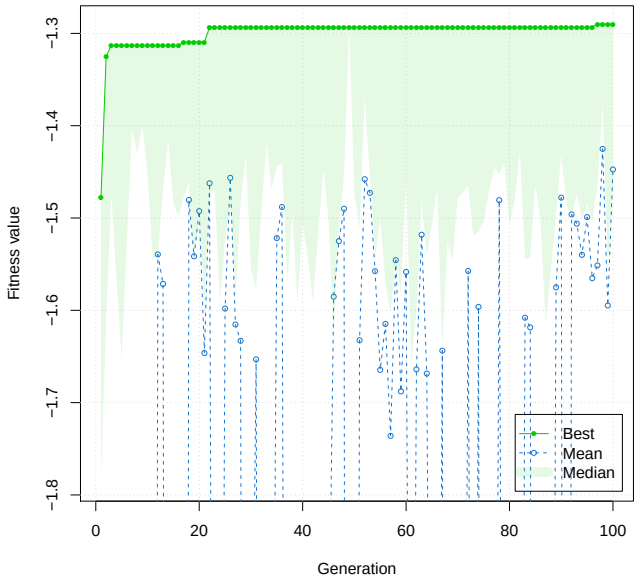

Fig. A.7 : Evolution of genetic algorithm for the subproblem 7

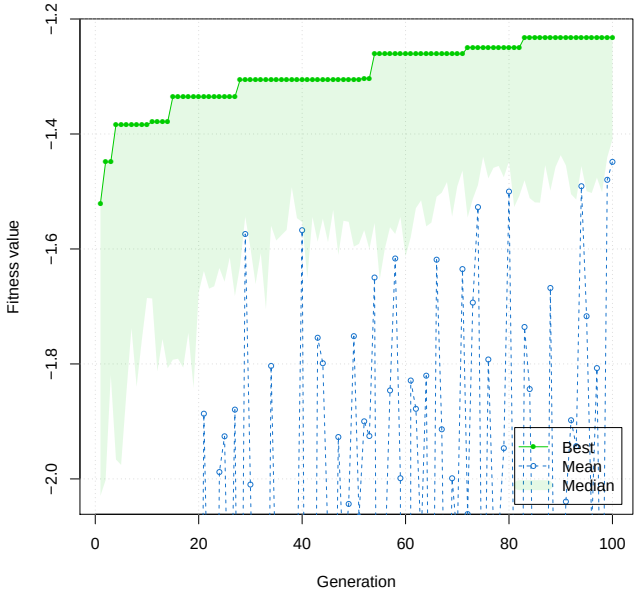

Fig. A.8 : Evolution of genetic algorithm for the subproblem 8

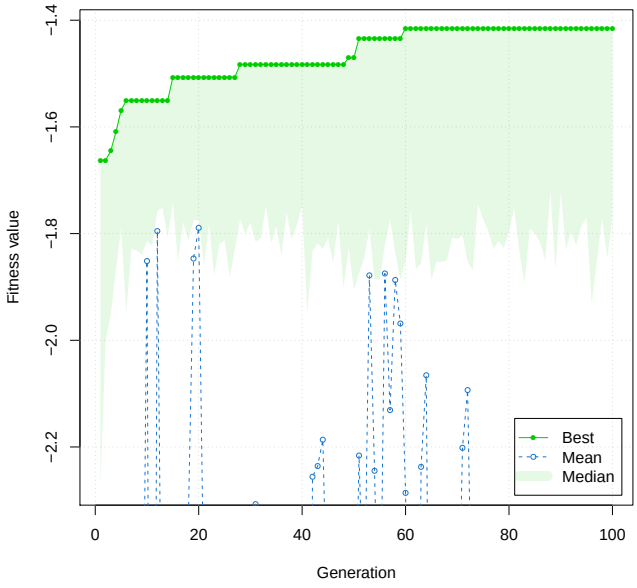

Fig. A.9 : Evolution of genetic algorithm for the subproblem 9

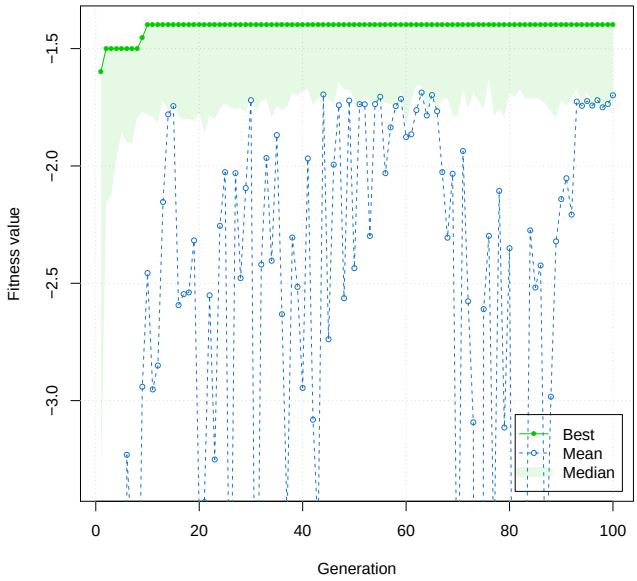

Fig. A.10 : Evolution of genetic algorithm for the subproblem 10

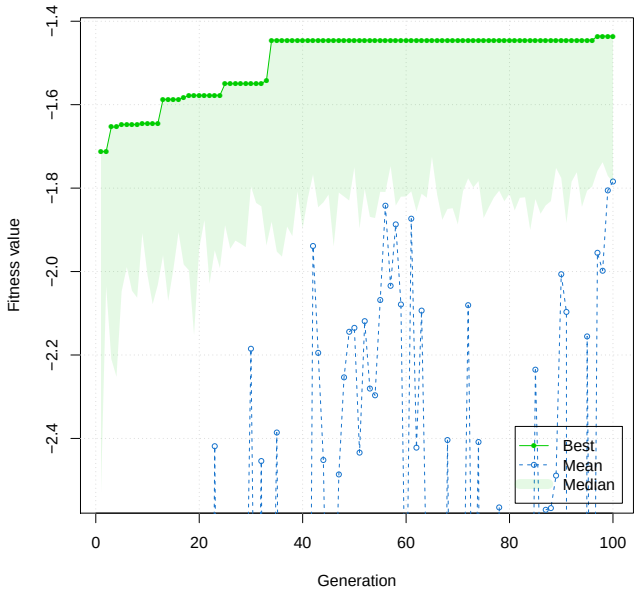

Fig. A.11 : Evolution of genetic algorithm for the subproblem 11

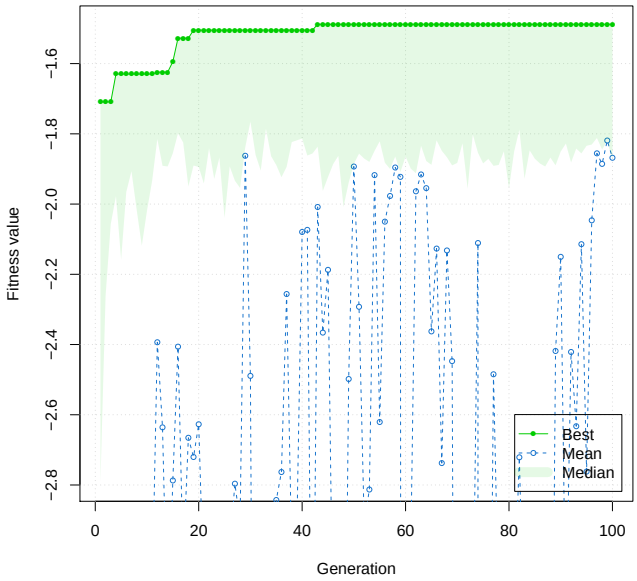

Fig. A.12 : Evolution of genetic algorithm for the subproblem 12

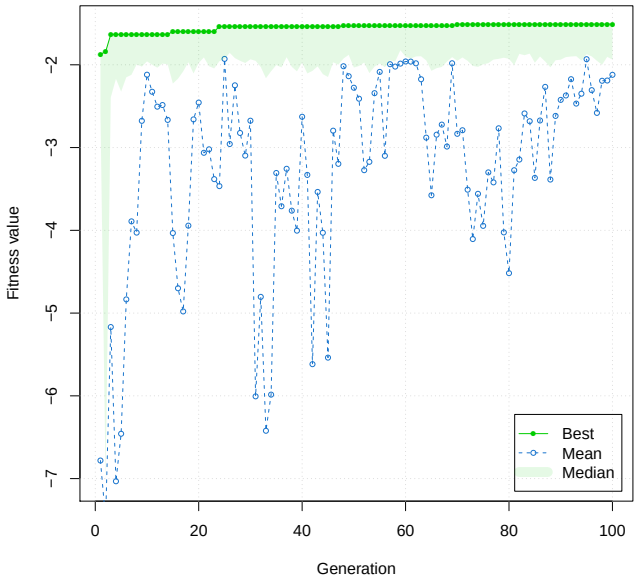

Fig. A.13 : Evolution of genetic algorithm for the subproblem 13

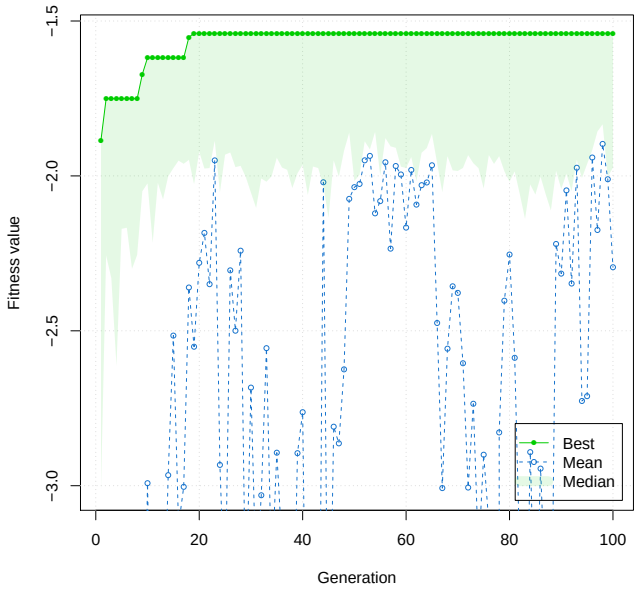

Fig. A.14 : Evolution of genetic algorithm for the subproblem 14

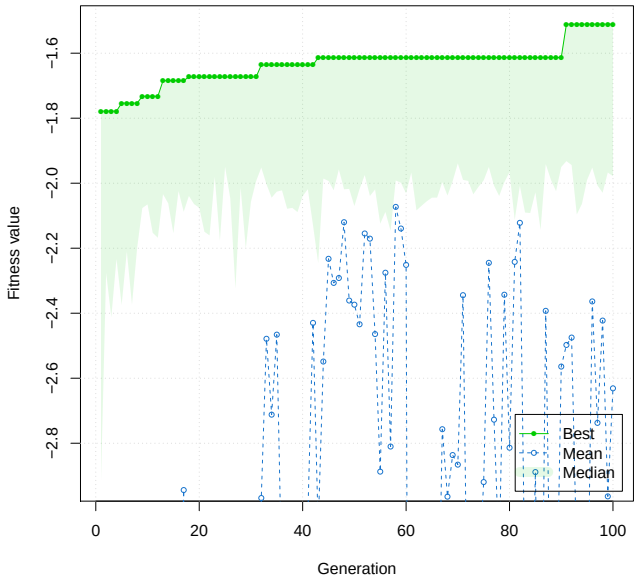

Fig. A.15 : Evolution of genetic algorithm for the subproblem 15

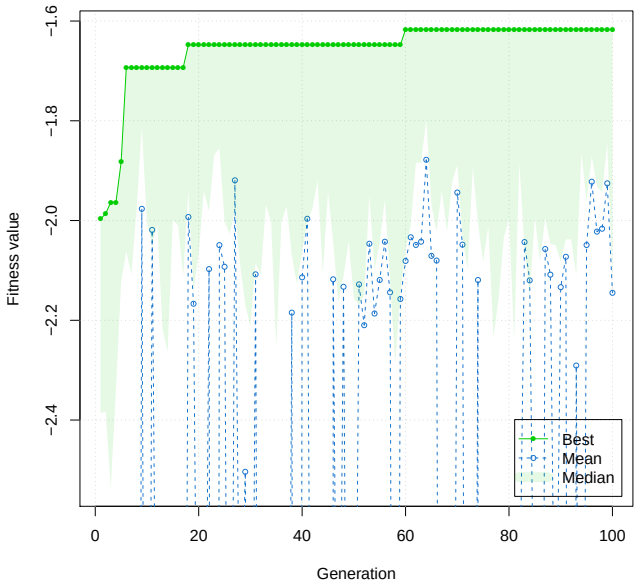

Fig. A.16 : Evolution of genetic algorithm for the subproblem 16

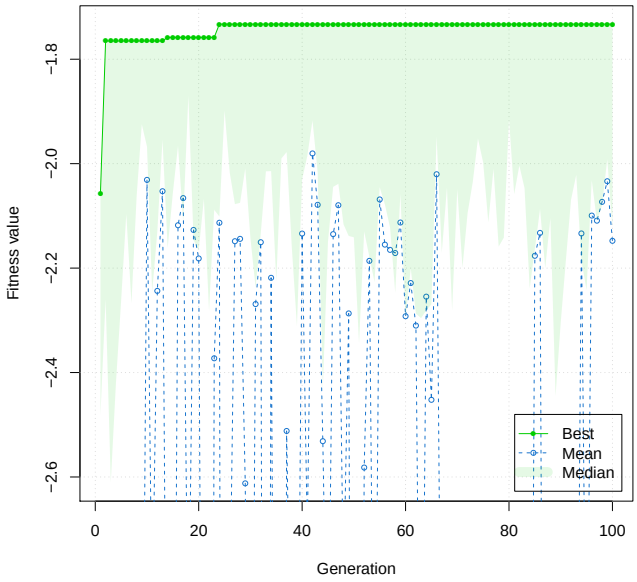

Fig. A.17 : Evolution of genetic algorithm for the subproblem 17

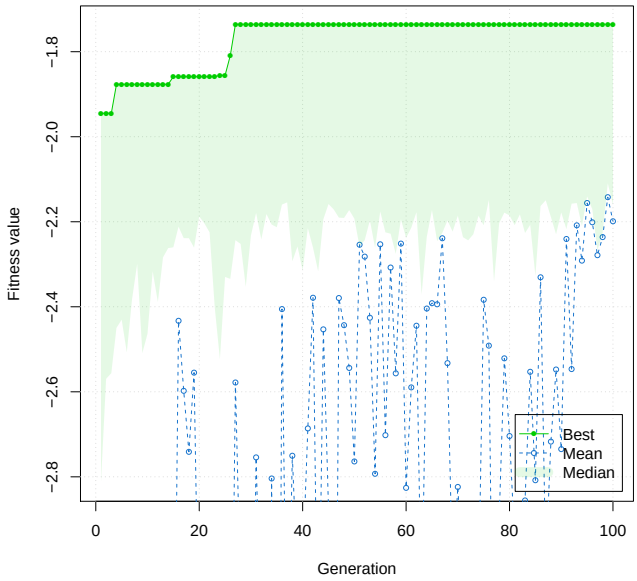

Fig. A.18 : Evolution of genetic algorithm for the subproblem 18

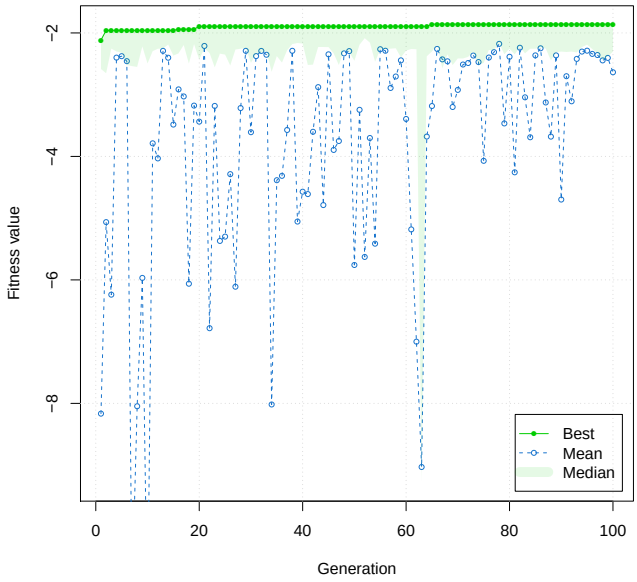

Fig. A.19 : Evolution of genetic algorithm for the subproblem 19

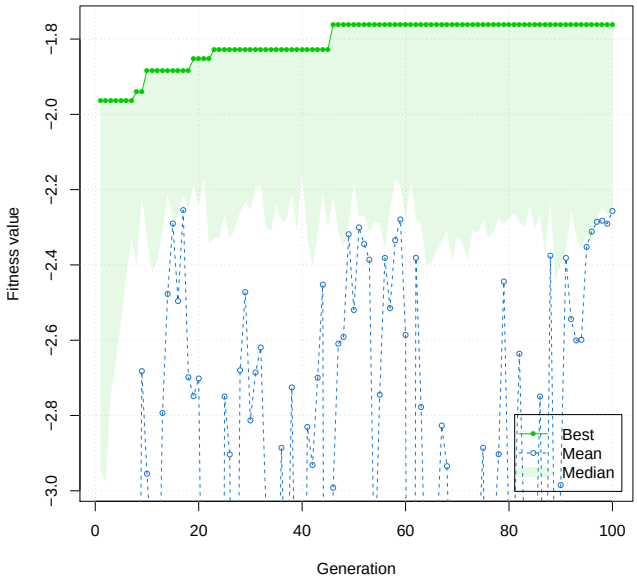

Fig. A.20 : Evolution of genetic algorithm for the subproblem 20

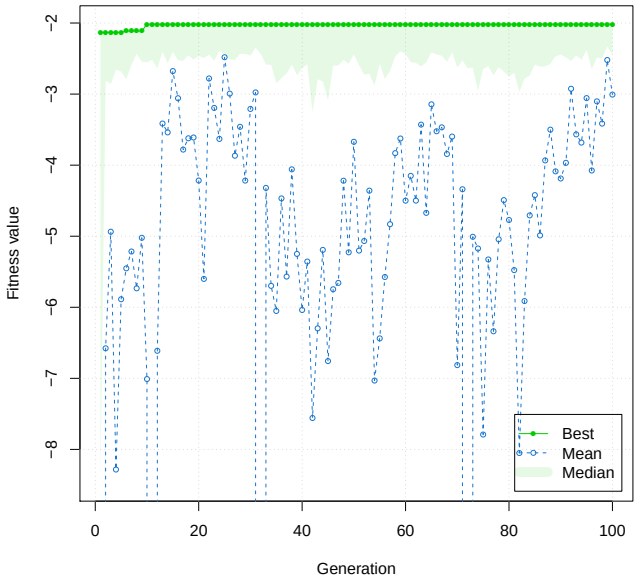

Fig. A.21 : Evolution of genetic algorithm for the subproblem 21

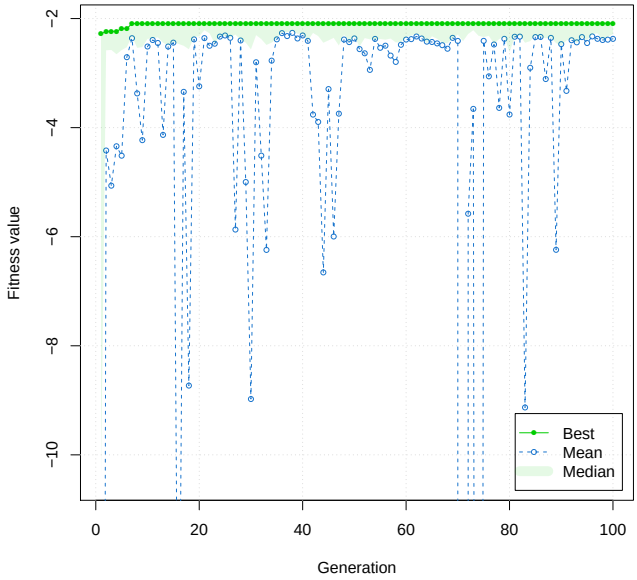

Fig. A.22 : Evolution of genetic algorithm for the subproblem 22

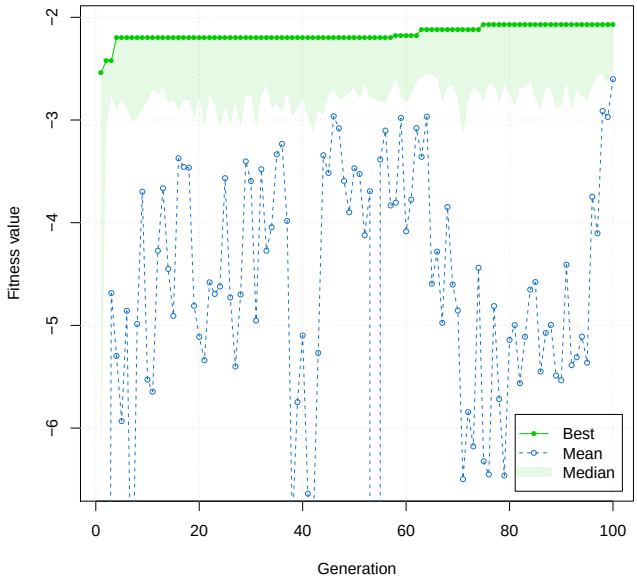

Fig. A.23 : Evolution of genetic algorithm for the subproblem 23

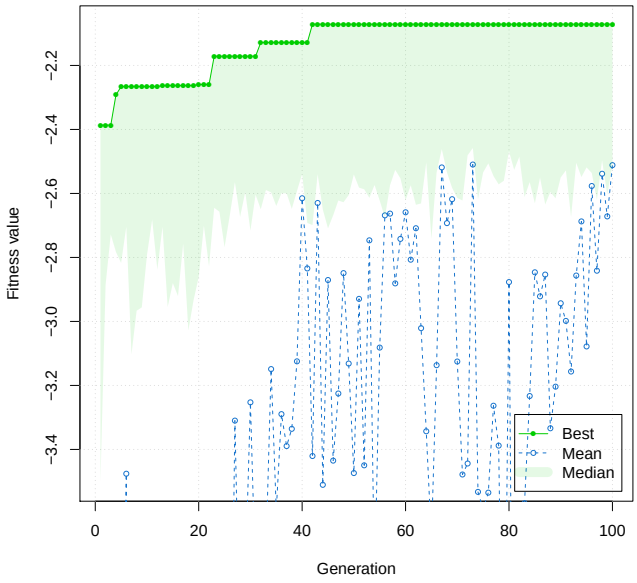

Fig. A.24 : Evolution of genetic algorithm for the subproblem 24